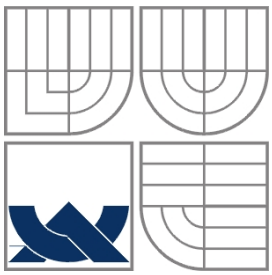


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ
ÚSTAV AUTOMATIZACE A INFORMATIKY

FACULTY OF MECHANICAL ENGINEERING
INSTITUTE OF AUTOMATION AND COMPUTER SCIENCE

OPTIMALIZACE SÍŤOVÉHO PŘEPÍNAČE POMOCÍ NEURONOVÉ SITĚ

NETWORK SWITCH OPTIMIZATION BY MEANS OF NEURAL NETWORK

DIPLOMOVÁ PRÁCE
DIPLOMA THESIS

AUTOR PRÁCE
AUTHOR
Bc. Jiří Lýsek

VEDOUCÍ PRÁCE
SUPERVISOR
doc. RNDr. Ing. Jiří Šťastný, CSc.

BRNO 2009

ZADÁNÍ ZÁVĚREČNÉ PRÁCE

(na místo tohoto listu vložte originál a nebo kopii zadání Vaš práce)

ABSTRAKT

Tato diplomová práce se zabývá řešením problému prioritního síťového přepínače, jehož model byl vytvořen v prostředí C++. Úloha optimálního přepínání je řešena pomocí několika umělých neuronových sítí, které jsou popsány, navzájem porovnány, a je vyhodnoceno, která se pro daný problém hodí lépe. Výsledkem práce je model přepínače a srovnání časové náročnosti při řešení optimalizačního problému pomocí umělé neuronové sítě.

Tato diplomová práce byla zpracována v rámci vědecko–výzkumného záměru MSM 0021630529 Inteligentní systémy v automatizaci.

ABSTRACT

This thesis deals with the problem of priority network switch, the model of which was developed in the C++ language. The traffic optimization task is solved by the use of several artificial neural networks, which are described, compared to each other and then evaluated which of them is more suitable for this task. The result of this work is a model of network switch and a comparison of computational time complexity of solving the optimization problem using the artificial neural network.

The thesis was developed in research project MSM 0021630529 Intelligent Systems in Automation.

KLÍČOVÁ SLOVA

neuronová síť, síťový prvek, optimalizace, Hopfieldova síť, Kohonenova síť, síť Competitive

KEYWORDS

neural network, network switch, optimization, Hopfield net, Kohonen net, Competitive net

Obsah:

Zadání závěrečné práce....5
Abstrakt....9
1 Úvod....13
2 Rozbor problému....15
2.1 Řízení síťového provozu....15
2.2 Využití datových sítí dnes a v minulosti....15
2.3 Stručný popis modelu....15
2.4 Optimalizace odeslaných paketů podle priorit....16
3 Použité řešení....19
3.1 Umělé neuronové sítě....19
3.1.1 Model umělého neuronu....20
3.1.2 Naučení sítě....21
3.2 Hopfieldova neuronová síť....23
3.2.1 Topologie....24
3.2.2 Interpretace v tomto problému....25
3.2.3 Naučení sítě....26
3.2.4 Průběh výpočtu konfiguračního vektoru....26
3.3 Kohonenova neuronová síť....28
3.3.1 Topologie....28
3.3.2 Interpretace v tomto problému....29
3.3.3 Naučení sítě....29
3.3.4 Průběh výpočtu....30
3.4 Neuronová síť Competitive....32
3.4.1 Topologie....32
3.4.2 Interpretace v tomto problému....32
3.4.3 Naučení sítě....33
3.4.4 Průběh výpočtu....33
3.5 Porovnání jednotlivých neuronových sítí....35
4 Popis naprogramovaného modelu....37
4.1 Zvolené vývojové prostředí....37
4.2 Výsledný program – popis prostředí....37
4.3 Jednotlivé programové jednotky....37
4.3.1 Hlavní správce simulace – třída Switch....38
4.3.2 Generátor paketů – třída PacketGenerator....40
4.3.3 Hlavní manažer dat – třída MainDataManager....40
4.3.4 Fronta paketů – třída PacketQueue....40
4.3.5 Obecný výpočet – třída Calculator....40
4.3.6 Výpočet Hopfieldovou neuronovou sítí – HopfieldComputing....41
4.3.7 Výpočet Kohonenovou neuronovou sítí – KohonenComputing....41
4.3.8 Výpočet neuronovou sítí Competitive – CopetitiveComputing....41
4.3.9 Obecná neuronová síť – třída NeuralNetwork....41
4.3.10 Výpočet Hopfieldovou sítí – třída HopfieldComputing....41
4.3.11 Výpočet Kohonenovou sítí – třída KohonenComputing....41
4.3.12 Výpočet sítí Competitive – třída CompetitiveComputing....41
4.3.13 Sériový výpočet – třída SerialComputing....41
4.4 Komunikace mezi objekty....41

4.4.1 Vektor priorit – PriorVector....41
4.4.2 Konfigurační vektor – ConfigVector....42
4.5 Doplňkové třídy nesouvisející přímo se simulací....42
5 Porovnání výsledků....43
5.1 Počty konfiguračních vzorů....43
5.2 Porovnání sítí mezi sebou....43
5.3 Porovnání Hopfieldovy sítě pro různé zatížení....46
5.4 Porovnání správnosti výsledků....48
6 Závěr....49
7 Seznam použité literatury....51

1 ÚVOD

V počítačových sítích jsou pro směřování dat využívány aktivní síťové prvky – přepínače. Jejich úkolem je zajistit, aby přijaté datové celky (pakety) od odesílatele byly přijaty a odeslány na správný výstup, kde by se měl nacházet jejich příjemce. V dnešní době, kdy se objemy datových přenosů stále zvětšují a rostou i přenosové rychlosti, je velmi důležité zabezpečit, aby data přijatá takovýmto aktivním prvkem, byla co nejrychleji odeslána k příjemci.

Pokud je navíc po těchto přepínačích požadováno, aby přijatá data byla zpracovávána i s přihlédnutím k jejich prioritě, kterou jim odesílatel přisoudil, může vzniknout problém s pořízením vhodného a laciného centrálního procesoru pro výpočet optimálního rozeslání dat. Z tohoto důvodu je vhodné nahradit jeden centrální výpočetní člen, větším počtem jednodušších paralelně zapojených funkčních bloků. Jednou z často používaných paralelních struktur jsou umělé neuronové sítě.

Neuronových sítí existuje mnoho druhů, ale ne každá se dá použít k takovému výpočtu, jehož vstupem by měly být hodnoty priorit ze vstupů přepínače a výsledkem by měl být rozvrh jak optimálně tyto pakety rozeslat. Proto je potřeba dobře zvolit typ neuronové sítě.

V této diplomové práci navazuji na předchozí práci na toto téma, která se věnovala tvorbě modelu přepínače řízeného Hopfieldovou neuronovou sítí. Tento model byl vytvořen v prostředí Matlab – Simulink. V mojí práci jsem podobný model vytvořil v prostředí C++ a rozšířil ho o další neuronové sítě – Kohonenovu a Competitive.

2 ROZBOR PROBLÉMU

2.1 Řízení síťového provozu

V počítačových sítích je k řízení provozu používán aktivní síťový prvek – přepínač (*switch*) [1]. Jeho funkce je poměrně jednoduchá. Má několik vstupních portů které jsou zároveň i výstupními. Na vstupech přijímá datové celky – tzv. *pakety* od připojených zařízení a podle adresy příjemce paketu, která se nachází v hlavičce paketu, odešle paket na zvolený výstupní port, kde se má nacházet příjemce paketu.

V práci se nebudu zabývat podrobnostmi řízení provozu v sítích, jelikož v modelu se nepracuje s nějakým konkrétním protokolem (např. TCP/IP), ale adresy odesílatelů/příjemců/ a obecně celé pakety, jsou jen symbolickým znázorněním toho, co doopravdy putuje v datových sítích. V této práci jde hlavně o zjištění, která neuronová síť je vhodná pro řízení síťového přepínače.

2.2 Využití datových sítí dnes a v minulosti

V nedávné minulosti se datové sítě využívaly, vzhledem ke své nízké rychlosti, hlavně pro přenos dat, které neměly příliš aktuální charakter – maily, soubory (ftp, www). Pro přenos například telefonních hovorů se používaly vlastní specifické sítě.

V dnešní době se zdá, že většina dat je digitalizována a dříve technologicky rozdílné sítě začínají konvergovat k jednomu typu datových sítí. To ovšem znamená, že se na síti zvětšuje objem dat a také to, že by bylo vhodné upřednostňovat určitá data před jinými.

Proto vznikla myšlenka přiřazování priorit k paketům. Abychom například mohli upřednostnit telefonní hovor před odesláním emailové zprávy, nebo upřednostnit průchod dat z nějakého video kanálu.

2.3 Stručný popis modelu

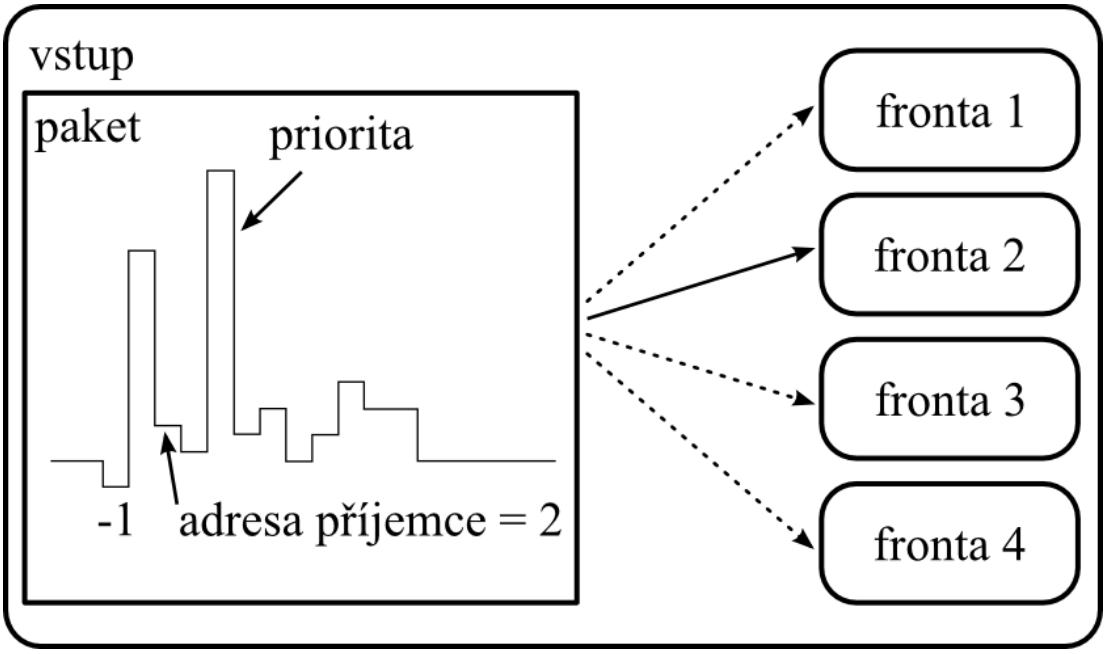
V práci, ze které vycházím [2] byl vytvořen model přepínače se čtyřmi porty. Příchozí paket na každý z těchto portů byl zařazen do jedné ze čtyř front. Každý vstup měl svoje čtyři fronty, podle toho jakou měl paket určenou adresu příjemce. Pakety byly generovány blokem PacketGenerator, který vytvářel sekvence dat podle následujícího popisu:

Pozice v datech	Význam	Hodnota
1	začátek paketu	-1
2	délka paketu	8 až 20
3	adresa příjemce	1 až 4
4	zdroj	1 až 4
5	priorita	0 až 98
6...a dále	data	1 až 5

Tabulka 2.1 Struktura paketu

Každý paket tedy začíná hodnotou -1, což identifikuje příchozí paket ve správci dat. Mezi pakety je určitá prodleva v rozmezí 5 až 20 simulačních cyklů. Priorita o hodnotě 0 znamená vysokou důležitost paketu a opačně hodnota 98 znamená malou důležitost. Další

důležitou hodnotou je adresa příjemce, podle které se paket zařadil do jedné ze 4 front.



Obr 2.1 Zpracování paketu na jednom ze vstupů

2.4 Optimalizace odeslaných paketů podle priorit

Pro numerický optimalizační výpočet lze použít následující postup. Zvolené neuronové sítě využívají jiný výpočetní postup, nicméně matice konfiguračních vzorů zůstává v podstatě stejná. A pro pochopení problematiky je vhodné se věnovat i tomuto řešení.

Přiřadíme-li každému paketu informaci o prioritě, potřebujeme potom v síťovém přepínači nějak tuto prioritu využít. Pokud jsme řekli, že lepší priorita má menší hodnotu, jde tedy o minimalizaci hodnoty, která vznikne násobením konfiguračního vektoru z matice konfiguračních vzorů $\boldsymbol{E}$ a vektoru priorit $\boldsymbol{p}$, který získáme přečtením priorit z front čekajících paketů uspořádaných do matice $\boldsymbol{P}$.

$$\boldsymbol{P}=\begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \\ p_{41} & p_{42} & p_{43} & p_{44} \end{bmatrix}$$

$$\vec{\boldsymbol{p}}=(p_{11}\ \ p_{12}\ \ p_{13}\ \ p_{14}\ \ p_{21}\ \ p_{22}\ \ p_{23}\ \ p_{24}\ \ p_{31}\ \ p_{32}\ \ p_{33}\ \ p_{34}\ \ p_{41}\ \ p_{42}\ \ p_{43}\ \ p_{44})$$

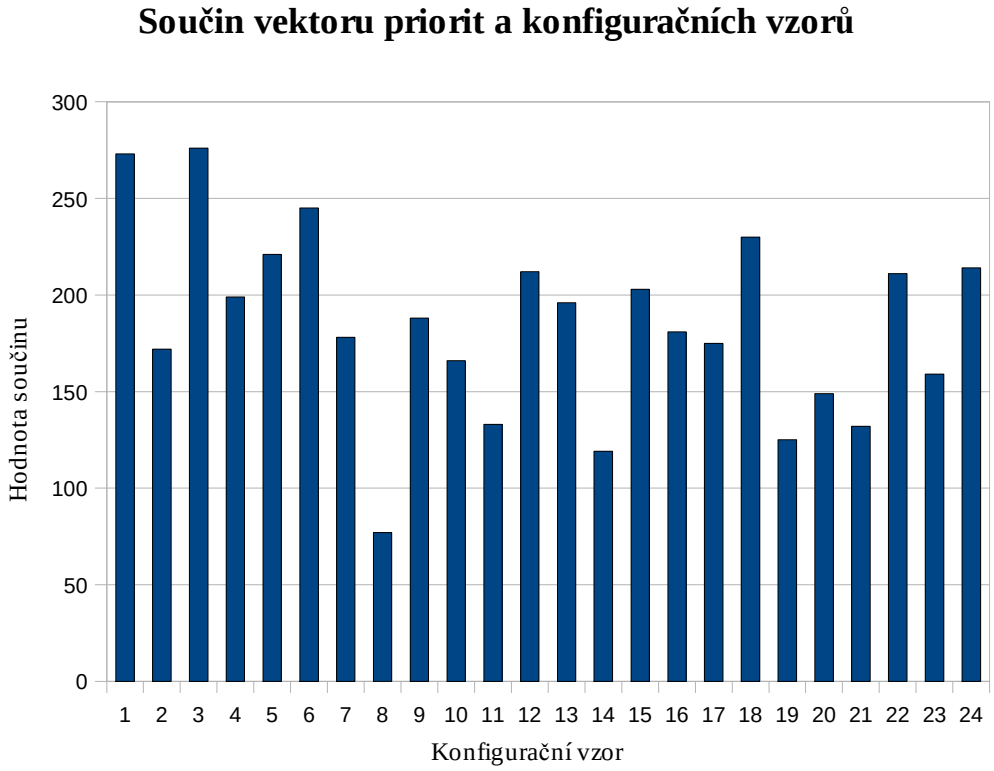
Matice $\boldsymbol{E}$ je soubor vzorových konfiguračních vektorů. Vektory jsou uspořádány do řádků. Pro přepínač se čtyřmi porty má matice $\boldsymbol{E}$ rozměry 24 × 16 (4! kombinací × 4 · 4 porty). Každá čtveřice na řádku matice $\boldsymbol{E}$**,** pak odpovídá jednomu portu, který má čtyři fronty pro čekající pakety. Pozice kde je v $\boldsymbol{E}_{ij} = 0$ znamená, že paket z tohoto zásobníku neodejde a pozice $\boldsymbol{E}_{ij} = 1$ znamená, že paket bude z daného zásobníku odeslán.

Minimalizace hodnoty pak vypadá takto:

$$\vec{\mathbf{sum}} = \vec{\boldsymbol{p}} \cdot (\boldsymbol{E})^T$$
$$\vec{\mathbf{sum}} = (s_1, s_2, \ldots, s_n)$$
$$min(\vec{\mathbf{sum}}) = s_i$$

Hodnota s_i je příslušná konfiguračnímu vzoru, který má nejmenší hodnotu součinu priorit a konfiguračního vzoru. Tento vzor je pak vybrán a pakety jsou ze zásobníků odeslány podle něj.

Příklad vypočtených hodnot je na obrázku *2.2*. Je na něm vidět, že nejmenší hodnotu součinu má vzor číslo 8 (odpovídá osmému řádku v matici ***E***). Hodnoty vektoru priorit jsou zvoleny takto: $\vec{\boldsymbol{p}} = (68, 24, 12, 80, 25, 36, 48, 15, 32, 50, 65, 78, 95, 18, 48, 45)$ a odpovídají hodnotám zvoleným v obrázku *3.7* z kapitoly *3.2.2*. K odeslání paketů tedy dojde z třetí fronty prvního vstupu, čtvrté fronty druhého vstupu, první fronty třetího vstupu a druhé fronty čtvrtého vstupu.



*Obr 2.2 Graf hodnot součinů vektoru priorit **p** a matice **E***

$$\boldsymbol{E} = \begin{pmatrix}
0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\
1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1
\end{pmatrix}$$

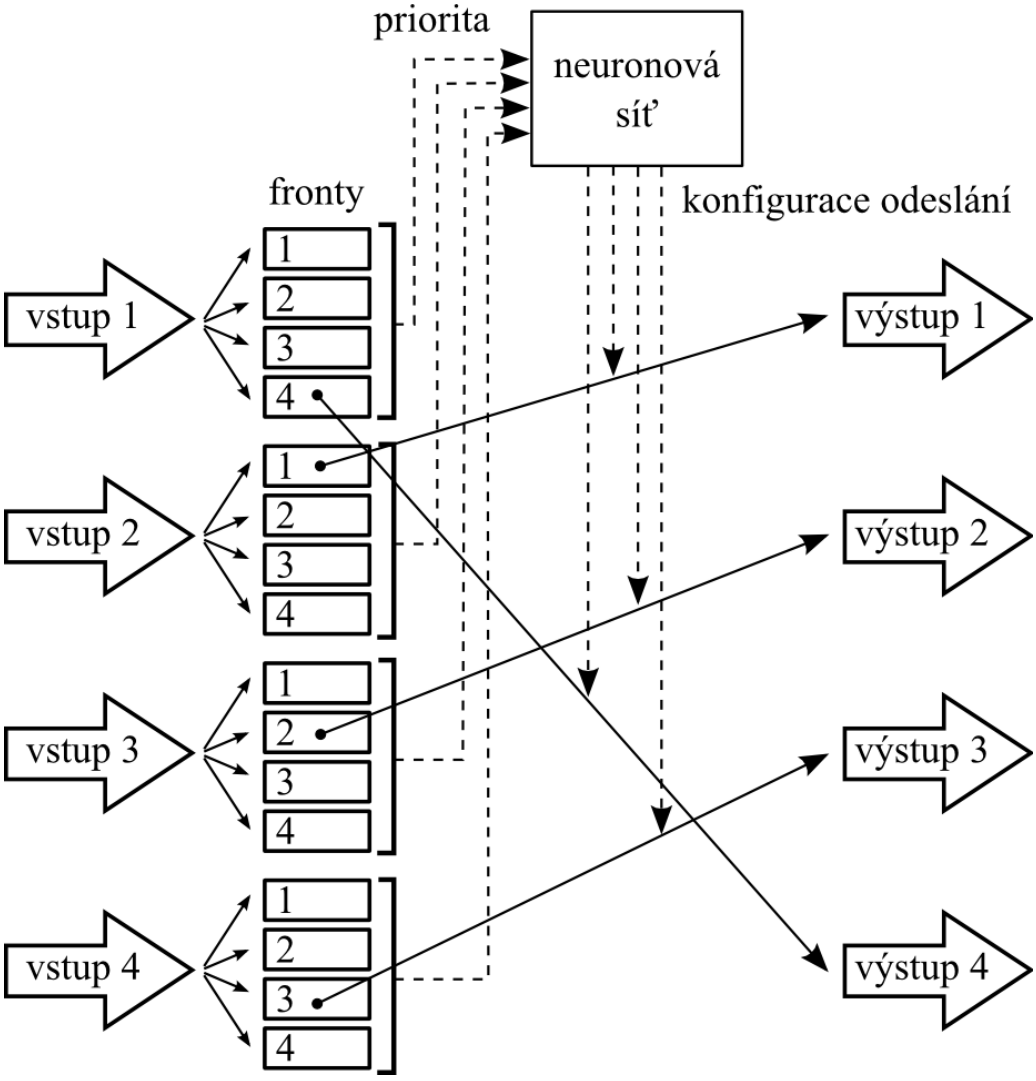
Matice ***E*** je v podstatě soubor vybraných kombinací, které jsou použitelné pro konfiguraci front na odeslání paketů. Důležité je, aby v každé čtveřici na jednom řádku byl vždy jen jeden příkaz k odeslání dat (1) a zároveň, aby v každé čtveřici byla vybraná fronta jiná, než ve všech ostatních čtveřicích.

Tuto matic lze vygenerovat pomocí jednoduchého algoritmu popsaného v práci [2] kapitola *3.2*.

3 POUŽITÉ ŘEŠENÍ

Jak bylo řečeno v úvodu, pro řešení optimalizačních problémů lze využít neuronovou síť, jejíž výhoda je její paralelní struktura. Pro řešení řízení aktivního síťového prvku byly zvoleny tyto druhy neuronových sítí:

- Hopfieldova neuronová síť
- Kohonenova neuronová síť
- Neuronová síť Competitive



Obr 3.1 Schéma přepínače

Na obrázku *3.1* je schéma síťového přepínače. Symbolicky jsou na něm znázorněny fronty, do kterých se zařazují pakety ze vstupních portů, podle adresy příjemce. Z front je potom odečtena priorita paketů a ta je zpracována v řídící neuronové síti. Po vypočtení vhodného konfiguračního vektoru je z front odeslán paket.

3.1 Umělé neuronové sítě

Nejprve se budu věnovat neuronovým sítím obecně. Umělé neuronové sítě (často

nazývané jen neuronové sítě) jsou matematickým modelem který se skládá z umělých neuronů a jejich vzájemných propojení. Obvyklé použití neuronové sítě je rozeznávání vzorů nebo hledání vztahů mezi vstupem a výstupem. Dají se však aplikovat i na jiné problémy.

Obvykle neuronové sítě rozdělujeme podle počtu vrstev, ve kterých jsou neurony:

- jednovrstvé
- dvouvrstvé
- třívrstvé

Více vrstev není potřeba, jelikož pomocí třívrstvé neuronové sítě jsme schopni řešit všechny problémy, které jsou lineárně nedělitelné (jednoduchou přímkou nelze oddělit data dvou různých kategorií). Obvykle se ve vícevrstvých sítích nazývá první vrstva vstupní, druhá vrstva (pokud je přítomna) skrytá a třetí (poslední) vrstva výstupní.

Neuronové sítě můžeme dělit i podle toho, jak v nich probíhá výpočet:

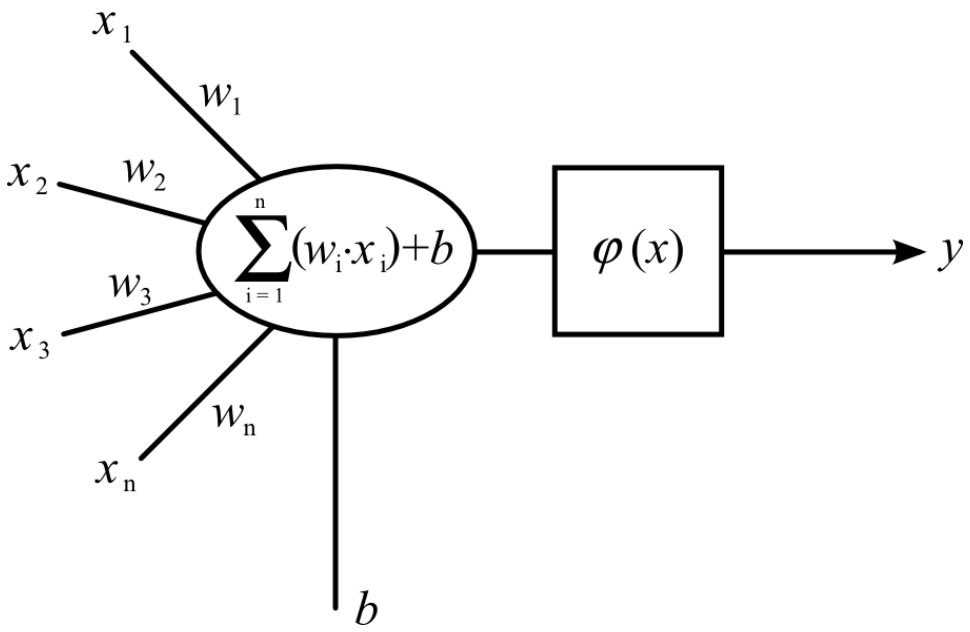
- dopředné
- zpětnovazební
- případně kombinované

U neuronové sítě také rozeznáváme 2 fáze její funkce:

- učení
- vybavování – aktivní stav

V průběhu učení probíhá nastavování parametrů sítě a potom je síť již normálně používána například k rozpoznávání. Existují však i sítě, které se učí průběžně a váhy si upravují v průběhu aktivní fáze.

3.1.1 Model umělého neuronu



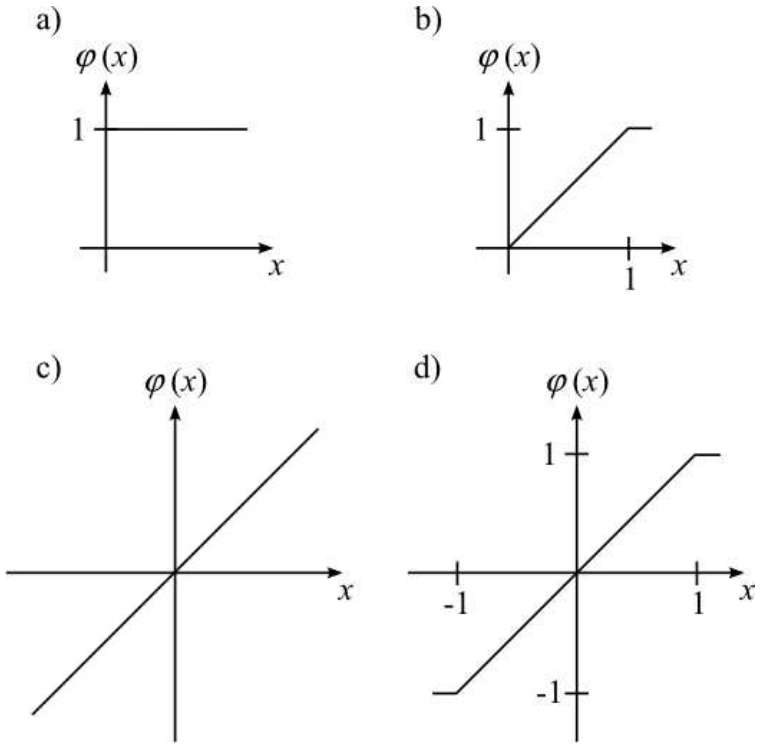
Obr. 3.2 Umělý neuron

Umělý neuron je matematickým modelem a neodpovídá tedy přesně lidskému neuronu. Skládá se ze dvou hlavních částí [3]. První část neuronu se stará o výpočet vážené sumy ze vstupů a druhá část se stará o výstup z neuronu, což je hodnota aktivační funkce $\varphi(x)$, kde x je hodnota vypočtené sumy v prvním bloku.

Vstupy neuronu x_i jsou obvykle výstupní hodnoty jiných neuronů. Mimo ně, má každý

neuron ještě vlastní hodnotu *b*, kterou nazýváme *bias – prahová hodnota. T*ato hodnota se nemění a je nastavena v průběhu učení sítě. Někdy se přiřazuje jako hodnota x_0 k vektoru vstupních hodnot $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$ s vahou $w_0 = 1$.

Aktivační funkce $\varphi(x)$ může mít různé tvary. Nejjednodušší je skoková (*Obr 3.3 a*) nebo lineární aktivační funkce (*Obr 3.3 c*), která může být nějak omezena (*Obr 3.3 b, d*). Složitější pak například *sigmoidální* aktivační funkce.



Obr 3.3 Některé aktivační funkce

3.1.2 Naučení sítě

Lze říci, že veškeré schopnosti neuronové sítě jsou dány pomocí topologie, vah, prahové hodnoty a typu aktivační funkce. Učící algoritmy jsou tedy konstruovány tak, aby měnily některé z jmenovaných parametrů a tím nastavily síť do takového stavu, který požadujeme.

Topologie
– je dána typem neuronové sítě a často se podle ní identifikuje o jakou neuronovou síť jde, nebo opačně, název sítě identifikuje topologii. Některé typy učení však topologii měnit mohou (např. učení pomocí genetických algoritmů).

Váhy
– nastavování vah je nejobvyklejší způsob učení neuronové sítě.

Prahová hodnota
– stejně jako váhy, i tato hodnota bývá často nastavována v průběhu učení.

Aktivační funkce
– je dána pevně a nemění se v průběhu učení. Je ale obvyklé, že na různých vrstvách sítě jsou jiné aktivační funkce.

Obvykle rozdělujeme učení na:

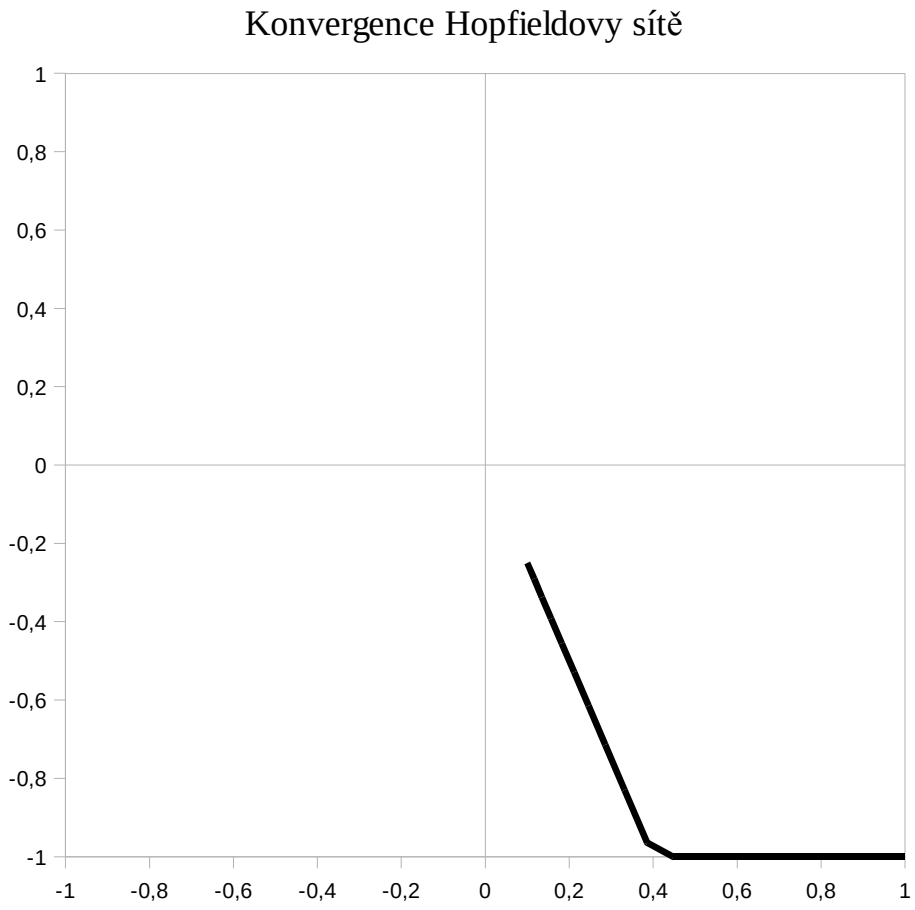
Učení s učitelem
: Neuronové síti jsou předkládány dvojce typu {vstupní vzor, žádaný výstup}. Po předložení vstupního vzoru je vypočten rozdíl a síť je upravena aby se výstup přiblížil žádanému.

Učení bez učitele
: Neuronové síti jsou předkládány jen hodnoty vstupu. Výstupní hodnoty jsou nám neznámé. Síť se sama zorganizuje.

Jiné metody
: Některé sítě jsou učeny jinými specifickými způsoby. Často jde o jednorázové nastavení vah pomocí nějakého výpočtu, nebo pro jednodušší a známé aplikace můžeme váhy nastavit ručně.

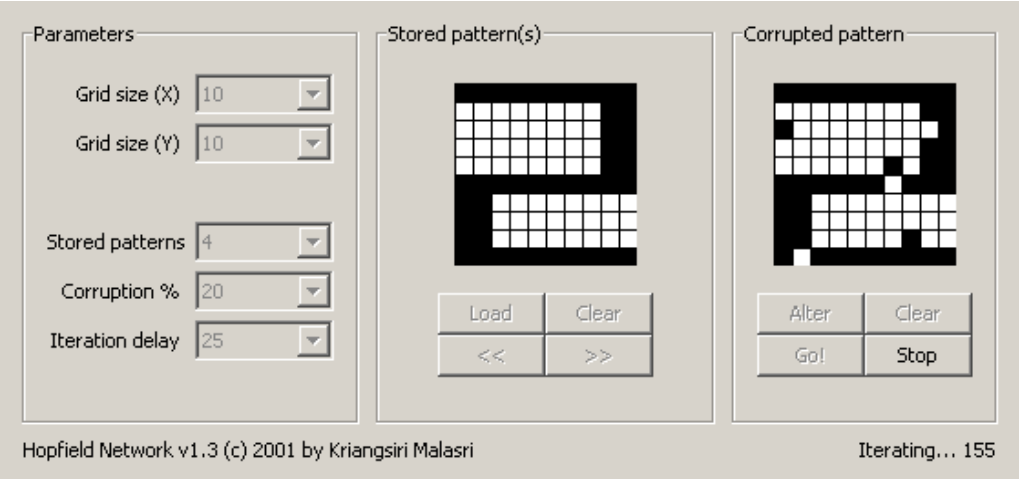
3.2 Hopfieldova neuronová síť

Jmenuje se podle svého tvůrce Johna Hopfielda (představena v roce 1982). Je to zpětnovazební síť s jednou vrstvou neuronů, která je často používaná jako asociativní paměť, protože je schopná podle vstupu, který je částečně znehodnocený nebo neúplný, vyvolat výstupní stav který jí byl během procesu učení předložen. Příklad takovéhoto použití je na obrázku *3.4*, kde je síti s dvěma neurony nastaven určitý vstup a po několika iteracích výpočtu přejde síť do naučeného stabilního stavu.



Obr. 3.4 Ukázka konvergence Hopfieldovy sítě z bodu [0,1; -0,25] do [1; -1]

Někdy je Hopfieldova síť používaná pro rozpoznávání textu (*OCR – Optical Character Recognition*) [4]. Její efektivnost v tomto použití není příliš velká, uvádí se schopnost zapamatovat si $0{,}14 \cdot n$ stavů [5], kde n je počet neuronů.

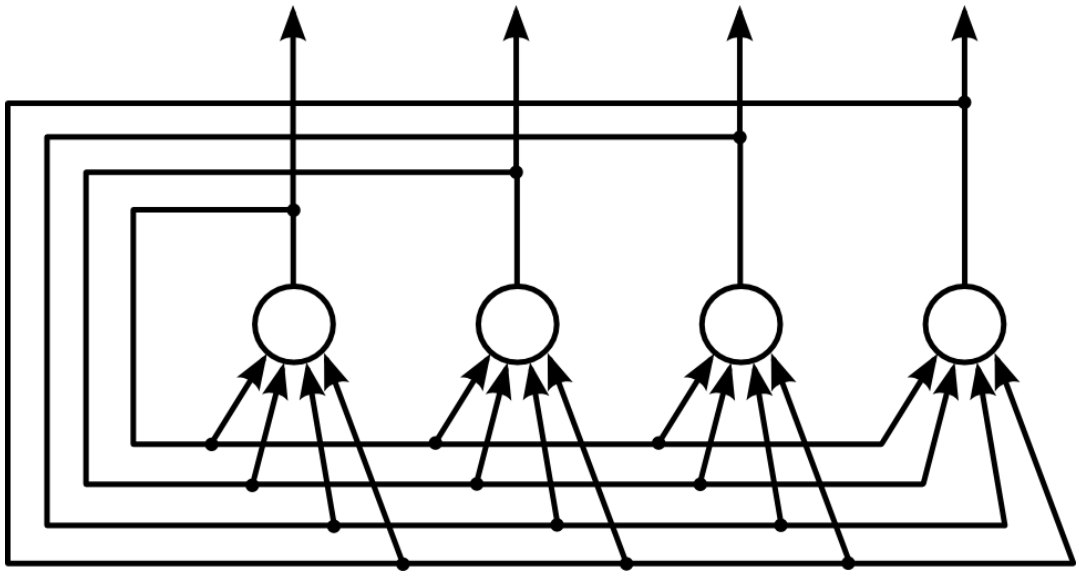


Obr. 3.5 Použití Hopfieldovy sítě pro OCR [4]

Tato síť lze také použít k řešení problémů optimalizačních [5]. Například problém obchodního cestujícího, problém N dam nebo problém N věží. Právě poslední jmenovaný se nejvíce podobá problému řízení síťového prvku. V podstatě jde o to, uspořádat na poli o velikosti N × N věže tak, aby se navzájem neohrožovaly (v každém řádku a každém sloupci je jen jedna věž). Navíc ještě uvažujeme cenu postavení věže na tomto poli (priorita paketu) a tuto hodnotu se snažíme minimalizovat.

3.2.1 Topologie

Jak již bylo řečeno, je to jednovrstvá zpětnovazební síť. Vstupem každého neuronu jsou výstupy všech ostatních, včetně jeho samého (některé zdroje však uvádí, že zpětná vazba neuronu sama na sebe není [6]). Tato síť nemá obvyklou vstupní a výstupní vrstvu, ale její vstup je nastaven jako počáteční hodnota výstupu všech neuronů. Výstupní hodnoty na konci výpočtu jsou opět získány jako výstupní hodnoty neuronů.



Obr. 3.6 Hopfieldova síť

3.2.2 Interpretace v tomto problému

Na neurony sítě bude nastaven normalizovaný vektor priorit v intervalu hodnot <-1, 1>. Kde priorita 98 odpovídá hodnotě -1 a priorita 0 odpovídá hodnotě 1. Pro přepočet je použita lineární závislost:

$$y = -\frac{1}{49} \cdot x + 1$$

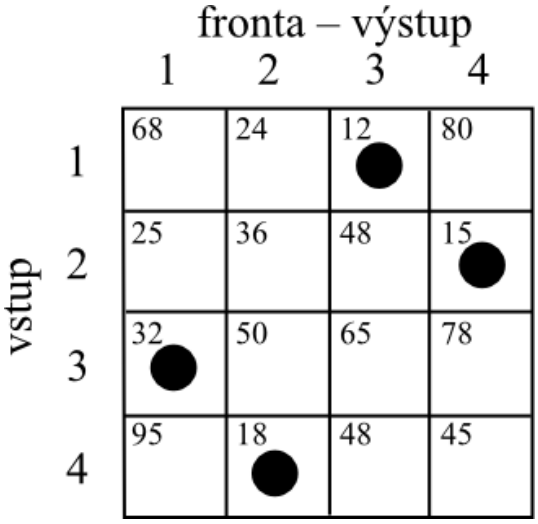
Síť potom provádí výpočet v několika iteracích, dokud nedojde ke konečnému výsledku..

Aktivační funkce
Pro výstup neuronu použijeme lineární závislost omezenou hodnotami -1 a 1:

$$\varphi(x) = \begin{cases} -1, & pro\ x \le -1 \\ x & \\ 1, & pro\ x \ge 1 \end{cases}$$

Výsledkem výpočtu neuronové sítě je vektor který se skládá z hodnot -1 a 1. Ten je interpretován tak, že hodnota 1 značí odeslání paketu z odpovídající fronty.

Pro přepínač se čtyřmi vstupy bude mít neuronová síť 16 neuronů (pro každou frontu jeden). Například výsledný vektor: $\vec{a} = (-1,-1,1,-1,-1,-1,-1,1,1,-1,-1,-1,-1,1,-1,-1)$ bude znamenat odeslání paketů z 3. fronty 1. vstupu na 3. výstup, 4. fronty 2. vstupu na 4. výstup atd. podle obrázku *3.7*.



Obr. 3.7 Ukázka interpretace konfiguračního vektoru, priority paketů jsou uvedeny v jednotlivých polích

3.2.3 Naučení sítě

Metod učení Hopfieldovy neuronové sítě je poměrně mnoho [7]. Ovšem většina popisovaných klasických i vylepšených metod není pro tuto úlohu vhodná. Abych dosáhl výsledky shodné s předchozím zpracováním [2], musel jsem přepsat do mého programu algoritmus z prostředí Matlab, jmenovitě příkaz *newhop* [3], který je použit pro tvorbu

Hopfieldových sítí v tomto prostředí a byl použit i v předchozí práci.

Algoritmus funkce *newhop* bere jako vstup matici ***t*** odpovídající transponované matici ***E*** z kapitoly *2.4* (místo hodnot 0 jsou hodnoty -1), ve které jsou vzory. Výstupem pak je matice ***W*** (matice vah) o rozměrech $n^2 \times n^2$ a jedno-sloupcová matice ***B*** (matice biasů) s n^2 řádky. Symbol *n* je počet vstupních portů.

3.2.4 Průběh výpočtu konfiguračního vektoru

Jelikož je síť zpětnovazební, je jasné že výpočet bude probíhat v několika iteracích. Zastavovacím kritériem výpočtu je stav, kdy výstup sítě je již ustálený a nemění se. Síť bývá popisována v synchronním (neurony se aktualizují sekvenčně a pro každou iteraci se používají výstupní hodnoty z minulé iterace) a asynchronním stavu (neurony se aktualizují náhodně a používají se okamžité hodnoty). V našem případě použijeme synchronní způsob práce:

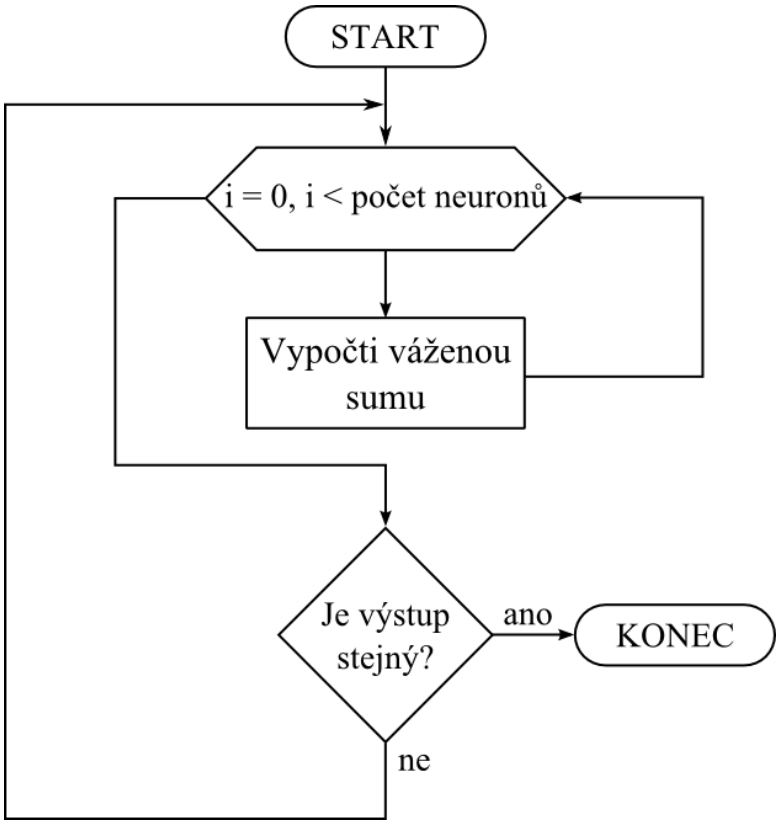
$$\vec{a}(0) = \textit{vektor vstupních hodnot}$$
$$\vec{a}(t) = \vec{w} \cdot \vec{a}(t-1) + \vec{b}$$

Zastavovacím kritériem je tedy: $\vec{a}(t) = \vec{a}(t-1)$

Vektor ***w*** odpovídá jednomu řádku z matice ***W*** a vektor ***b*** odpovídá sloupcové matici ***B***.

K získání výsledku je zapotřebí většího a dopředu neznámého počtu iterací. Navíc jsem zjistil, že pokud je větší množství vstupů do neuronové sítě rozmístěno blízko k jedné z krajních hodnot (-1 nebo 1), potom Hopfieldově síti zabere vybavení výsledku poměrně mnoho (i stovky) iterací. Tím se doba výpočtu poměrně prodlužuje.

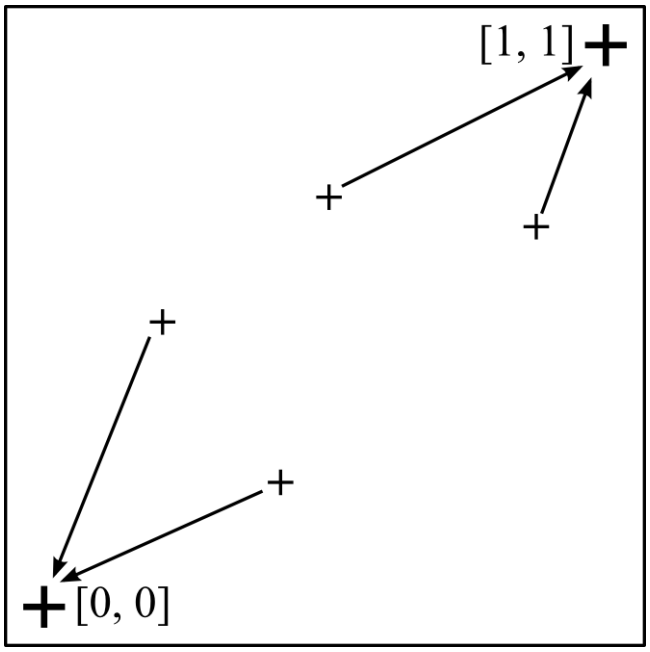
Výhodou na druhou stranu je, že počet neuronů roste jen s druhou mocninou v závislosti na počtu vstupních portů přepínače.



Obr 3.8 Vývojový diagram výpočtu pomocí Hopfieldovy sítě

3.3 Kohonenova neuronová síť

Tato síť je primárně určena k rozdělování předkládaných vzorů do určitých skupin. Používá k tomu hodnocení Euklidovské vzdálenosti mezi vzorem a neuronem, kterému je jeho pozice nastavena během procesu učení prostřednictvím vah. Vymyslel ji Finský profesor Teuvo Kohonen v osmdesátých letech [8]. V pravém slova smyslu nejde úplně přesně o neuronovou síť protože neurony nerealizují klasický model umělého neuronu (kapitola *3.1.1*), ale hodnotí vzdálenost vstupu od nějaké naučené znalosti. Navíc výsledný vektor, který vznikne po výpočtu je potřeba ještě zpracovat (vybrat nejmenší hodnotu – nejbližší vzor).

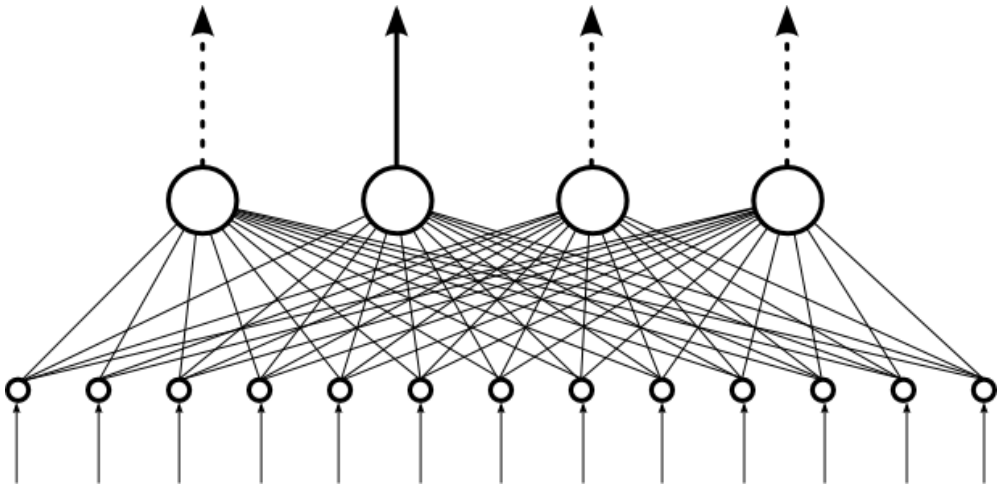


Obr. 3.9 Funkce Kohonenovy sítě

Na obrázku *3.9* je Kohonenova síť pracující v 2D prostoru (má dva vstupní neurony), tato síť byla naučena na 2 souřadnice ([0, 0], [1, 1] – dva výstupní neurony). Pokud síti předložíme bod, ležící blíž k jedné z těchto souřadnic než k druhé, síť zvolí jako „vítěze“ bližší bod.

3.3.1 Topologie

Tato síť je dopředná a dvouvrstvá. Skládá se z vrstvy vstupní a vrstvy výstupní. Každý neuron na výstupní vrstvě je propojen se všemi vstupy.



Obr. 3.10 Kohonenova síť

3.3.2 Interpretace v tomto problému

Na vstup sítě bude přiveden aktuální stav priorit na frontách přepínače, které budou normalizovány do intervalu <0, 1>, kde 0 odpovídá hodnotě priority 0 a 1 odpovídá hodnotě priority 98. Použije se tedy lineární závislosti:

$$y = \frac{1}{98} \cdot x$$

Neuronová síť by měla vybrat nejbližší konfigurační vzor. Pokud bude mít přepínač čtyři vstupy, budeme pracovat s šestnácti prioritami a budeme tedy pracovat v šestnácti rozměrném prostoru. Síť bude vybírat jako vítězný ten neuron, který je nejblíž vstupnímu vektoru ***p***. A tedy na výstupním neuronu má nejmenší hodnotu.

Pro přepínač se čtyřmi vstupy budeme mít tedy 16 vstupních neuronů a podle matice ***E*** z kapitoly *3.3.3* bude mít druhá vrstva 24 neuronů.

Aktivační funkce

Pro obě vrstvy není výstupní hodnota po výpočtu vážené sumy upravována.

$$\varphi_1(x) = \varphi_2(x) = x$$

3.3.3 Naučení sítě

Síť je obvykle učena bez učitele. Předkládané vzory jsou porovnány podle vzdálenosti na každém neuronu. Pro neuron, který je nejblíže, je provedena úprava vah, tak aby se přiblížily ke vstupnímu vzoru. Obvykle je upraveno i určité okolí vybraného neuronu [9].

V našem případě jsou váhy pro každý výstupní neuron nastaveny napevno podle konfiguračních vzorů ***E***:

$$\boldsymbol{E} = \begin{pmatrix}
1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \\
1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \\
1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 \\
1 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 \\
1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 \\
1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 \\
1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 \\
1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 \\
1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \\
0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \\
1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 \\
0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 \\
1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 \\
1 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 \\
1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 \\
0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 \\
1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 \\
0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 \\
1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\
1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\
1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\
0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\
1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\
0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0
\end{pmatrix}$$

Je vidět, že matice konfiguračních vzorů má inverzní hodnoty oproti matici ze sériového numerického výpočtu. To je odůvodněno tím, že řádky znázorňují body v šestnácti–rozměrném prostoru a je potřeba aby pakety s vyšší důležitostí (priority blíž k nule) a s menší důležitostí (priorita blíž k jedné) byly přiřazeny ke správnému vzoru.

Výsledkem výpočtu je potom jeden z řádků matice ***E*** získaný z vektoru vah vítězného neuronu. Ten je interpretován takto: pokud je $\boldsymbol{E}_{ij} = 0$, je z dané fronty odeslán paket podobně jako v kapitole *3.2.2* pro hodnotu 1. Jinak paket odeslán z fronty není.

3.3.4 Průběh výpočtu

Výpočet probíhá porovnáním vzdálenosti vstupní souřadnice se souřadnicemi všech výstupních neuronů. Ten neuron, který je nejblíže předloženému vstupu je označen jako vítěz a je zvolen jako výstup sítě.

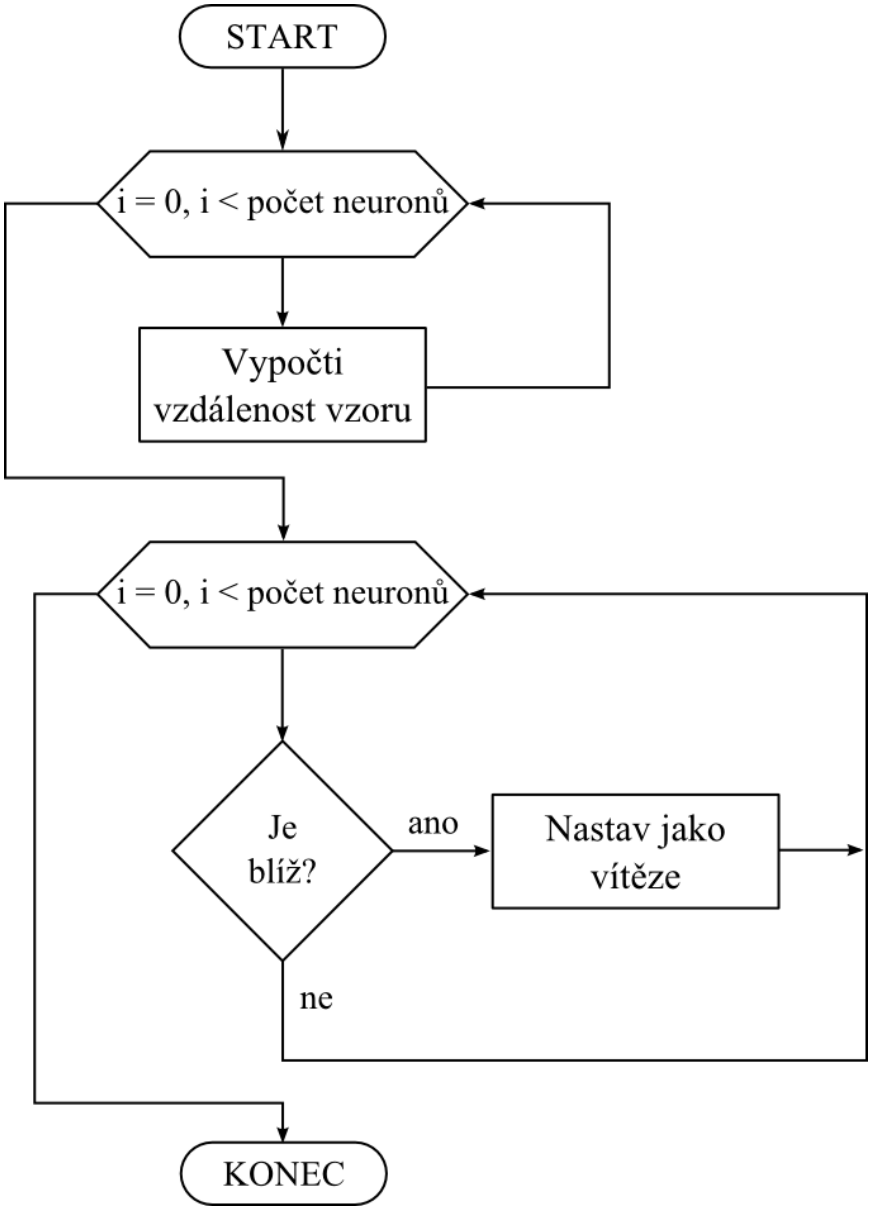
Vzdálenost vstupu P a určitého stavu Q_i, který odpovídá řádku v matici ***E***, je hodnocena podle Euklidovské vzdálenosti v obecně n-dimenzionálním prostoru:

$$P = \{p_{1,} p_{2,} \dots , p_n\}$$
$$Q_i = \{q_{i1}, q_{i2}, \dots , q_{in}\}$$
$$D_i = \sqrt{(p_{i1} - q_{i1})^2 + (p_{i2} - q_{i2})^2 + \dots + (p_{in} - q_{in})^2}$$

Souřadnice bodu *P* jsou vstupní hodnoty, které jsou nastaveny na vstupní vrstvu. Souřadnice bodu *Q* jsou uloženy ve vektoru vah pro daný výstupní neuron. Jako výsledný neuron použijeme ten, který má ve výpočtu nejmenší hodnotu *D*.

Výhodou této sítě je, že je dopředná a výsledek z ní získáme tím, že po průchodu vstupních hodnot touto sítí pouze porovnáme vzájemně výstupní hodnoty a vybereme tu nejmenší, ta identifikuje neuron, jehož váhy skrývají správný konfigurační vektor. Výpočet je tedy velmi krátký.

Nevýhodou je, že se zvětšujícím se počtem vstupních portů přepínače roste počet neuronů ve výstupní vrstvě s faktoriálem tohoto počtu a počet vstupních neuronů je vždy roven druhé mocnině tohoto počtu. Počet výpočtů v této síti se tedy s větším počten neuronů značně zvětšuje.



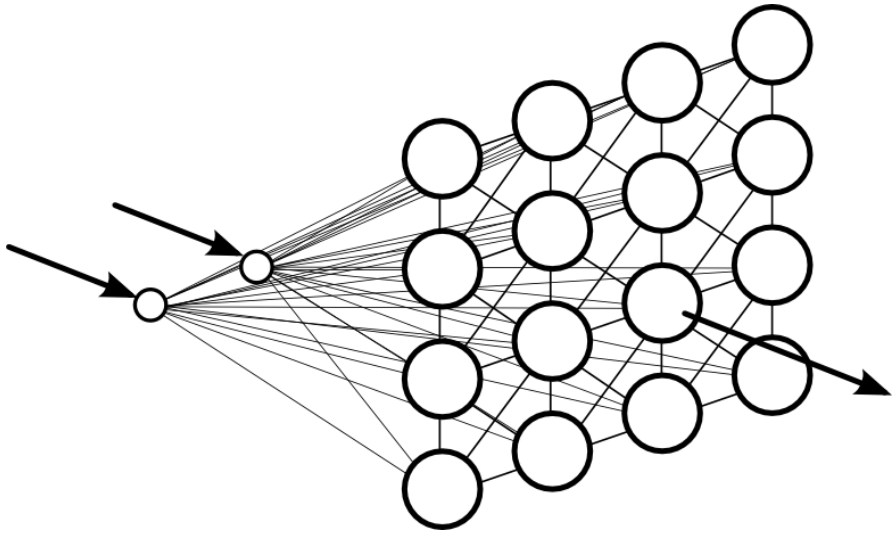
Obr 3.11 Vývojový diagram výpočtu pomocí Kohonenovy sítě

3.4 Neuronová síť Competitive

Je to dvouvrstvá neuronová síť, která kombinuje dopředné a rekurentní zapojení neuronů [10]. Používá se většinou k rozdělování vzorů do kategorií, podobně jako Kohonenova neuronová síť.

3.4.1 Topologie

Má dvě vrstvy. Všechny neurony na druhé vrstvě jsou propojeny se všemi neurony na vstupní vrstvě jako u Kohonenovy sítě. Dále jsou pak propojeny všechny neurony na druhé vrstvě navzájem, podobně jako u Hopfieldovy sítě.



Obr 3.12 Neuronová síť Competitive

3.4.2 Interpretace v tomto problému

Na vstup sítě bude přiveden aktuální stav priorit na frontách přepínače, které budou normalizovány do intervalu <0, 1>, kde 0 odpovídá hodnotě priority 0 a 1 odpovídá hodnotě priority 98. Použije se tedy lineární závislosti:

$$y = \frac{1}{98} \cdot x$$

Podobně jako u Kohonenovy neuronové sítě proběhne výpočet výstupní hodnoty pro každý neuron na druhé vrstvě. Tím ale práce neuronové sítě nekončí a začne rekurentní výpočet na druhé vrstvě s hodnotami z první fáze.

Pro přepínač se čtyřmi vstupy budeme mít tedy 16 vstupních neuronů a podle matice ***E*** z kapitoly *3.3.3* bude mít druhá vrstva 24 neuronů.

Aktivační funkce
V první vrstvě je výstupem nastavená hodnota bez omezení a v druhé vrstvě je hodnota výstupu omezena zezdola nulou.

$$\varphi_1(x) = x$$

$$\varphi_2(x) = \begin{cases} 0, & pro\ x < 0 \\ x \end{cases}$$

3.4.3 Naučení sítě

Naučení sítě probíhá jednorázově. Váhy spojení mezi vstupní a výstupní vrstvou jsou nastaveny podle matice vzorů, která je stejná jako u Kohonenovy neuronové sítě v kapitole *3.3.3*.

Váhy spojení na výstupní vrstvě jsou také nastaveny jednorázově a to tak, aby po několika iteracích výpočtu zůstal aktivní pouze neuron s největší počáteční hodnotou výstupu z první fáze výpočtu. Váhy jsou nastaveny takto [11]:

$$w_{ij} = 1 \qquad pro\ i = j$$
$$0 < w_{ij} < \frac{1}{n-1} \qquad jinak$$

Hodnota *n* je počet neuronů výstupní vrstvy.

3.4.4 Průběh výpočtu

V první části výpočtu je vstupní stav ***p*** násoben s nastavenými váhami ***q*** každého neuronu, které odpovídají řádkům matice ***E***. Tím vznikne na neuronu, který bude odpovídat nejlepší konfiguraci největší hodnota výstupu. V druhé části výpočtu pak neurony na druhé vrstvě „soutěží“ o to, který se stane výstupním neuronem sítě. V některých případech (například když jsou všechny vstupy nastaveny na stejnou hodnotu) se může stát, že nebude vybrán ani jeden neuron. V takovém případě můžeme vybrat náhodnou konfiguraci.

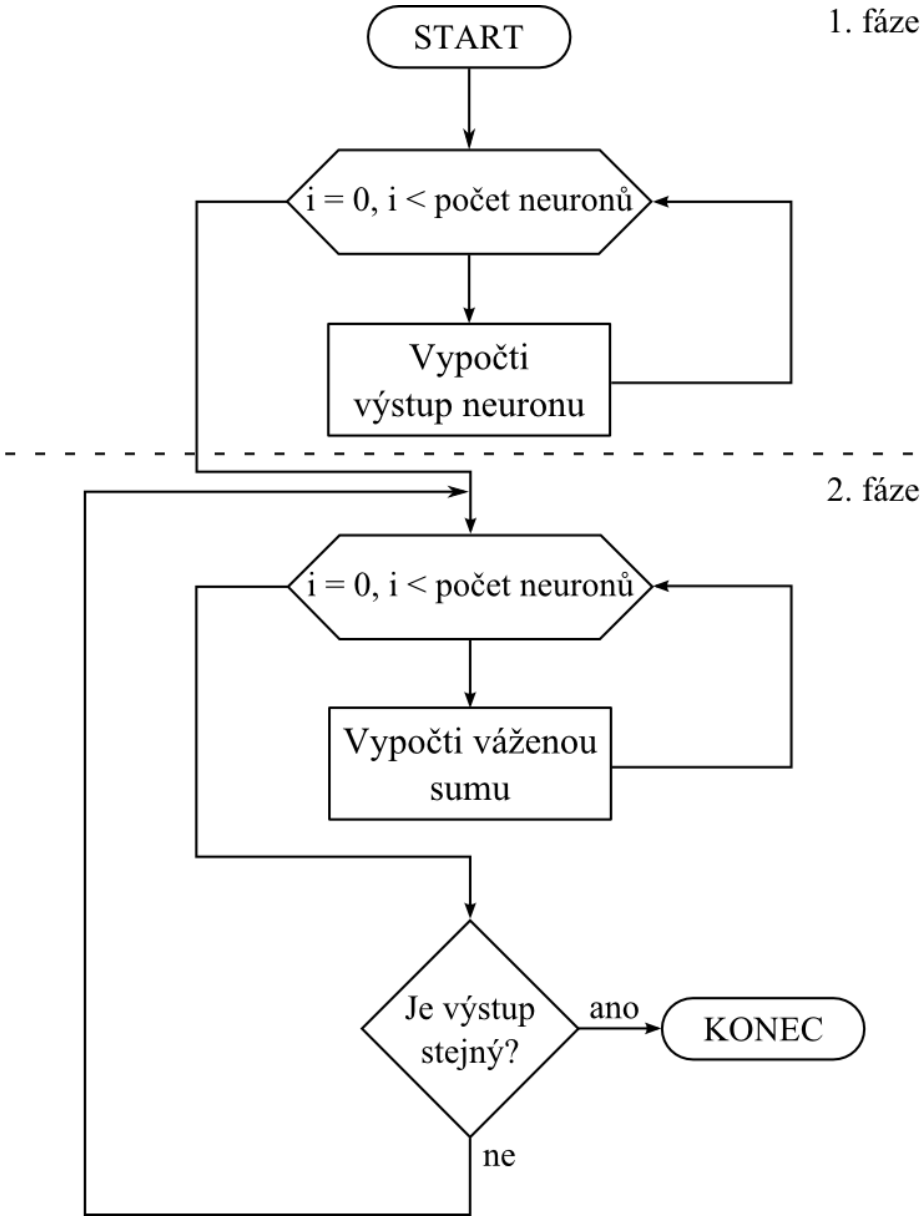
$$\vec{\boldsymbol{p}} = \{p_1, p_2, \ldots, p_n\}$$
$$\vec{\boldsymbol{q}}_{\boldsymbol{i}} = \{q_{i1}, q_{i2}, \ldots, q_{in}\}$$
$$a_i = p_1 \cdot q_{i1} + p_2 \cdot q_{i2} + \ldots + p_n \cdot q_{in}$$

$$\vec{\boldsymbol{a}}(0) = \{a_1, a_2, \ldots, a_n\}$$
$$\vec{\boldsymbol{a}}(t) = \vec{\boldsymbol{w}} \cdot \vec{\boldsymbol{a}}(t-1)$$

Hodnoty ***p*** jsou vstupní hodnoty, které jsou nastaveny na vstupní vrstvu. Hodnoty ***q*** jsou uloženy ve vektoru vah pro daný výstupní neuron. Po vypočtení hodnoty $\vec{\boldsymbol{a}}(0)$ se provádí rekurentní výpočet na druhé vrstvě tak dlouho, dokud neplatí: $\vec{\boldsymbol{a}}(t) = \vec{\boldsymbol{a}}(t-1)$

Vektor ***w*** odpovídá nastavení vah v kapitole *3.4.3*.

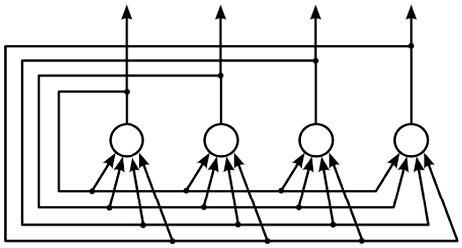
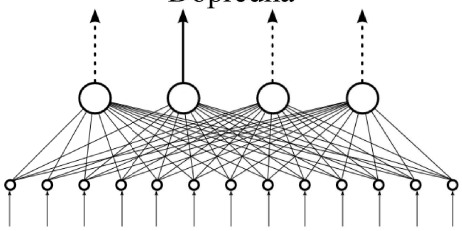
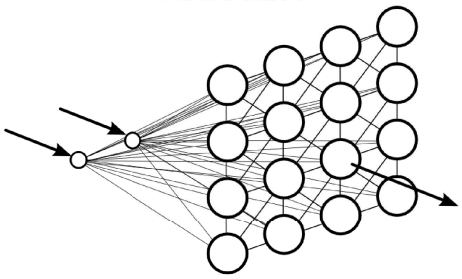
Tato neuronová síť bohužel spojuje obě nevýhody předchozích sítí: je to její rekurentní charakter podobný Hopfieldově neuronové síti v druhé vrstvě a růst počtu neuronů pro zvětšující se počet vstupních portů stejný jako u Kohonenovy sítě.



Obr 3.13 Vývojový diagram výpočtu pomocí sítě Competetive

3.5 Porovnání jednotlivých neuronových sítí

Pro porovnání jsem sestavil tabulku neuronových sítí, které jsem použil. Ve sloupci Počet neuronů je prezentován počet neuronů na první a druhé vrstvě pomocí součtu: první vrstva + druhá vrstva. Hodnota *n* je počet vstupních portů přepínače.

Název	Topologie	Počet neuronů	Aktivační funkce
Hopfieldova	Rekurentní 	$n^2 + 0$	$\varphi(x)=\begin{cases}-1, & pro\ x \le -1\\ x & \\ 1, & pro\ x \ge 1\end{cases}$
Kohonenova	Dopředná 	$n^2 + n!$	$\varphi_1(x)=\varphi_2(x)=x$
Competitive	Kombinace 	$n^2 + n!$	$\varphi_1(x)=x$ $\varphi_2(x)=\begin{cases}0, & pro\ x<0\\ x & \end{cases}$

Tabulka 3.1 Porovnání neuronových sítí

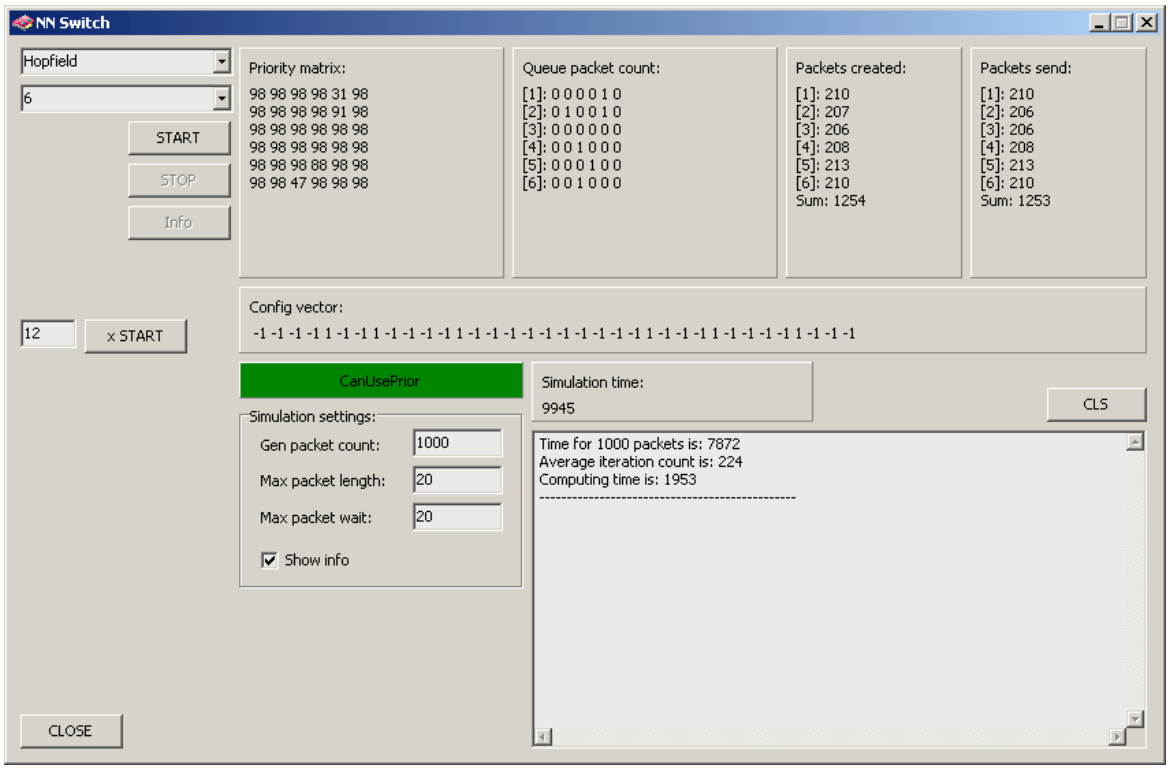
4 POPIS NAPROGRAMOVANÉHO MODELU

4.1 Zvolené vývojové prostředí

Pro implementaci modelu jsem zvolil jazyk C++ a prostředí Borland C++ Builder ve verzi Explorer, která je volně dostupná [12]. Výhodou tohoto prostředí je knihovna *VCL* (*Visual Component Library*). Ta se stará o práci s prvky pro interakci s programem jako jsou tlačítka apod..

4.2 Výsledný program – popis prostředí

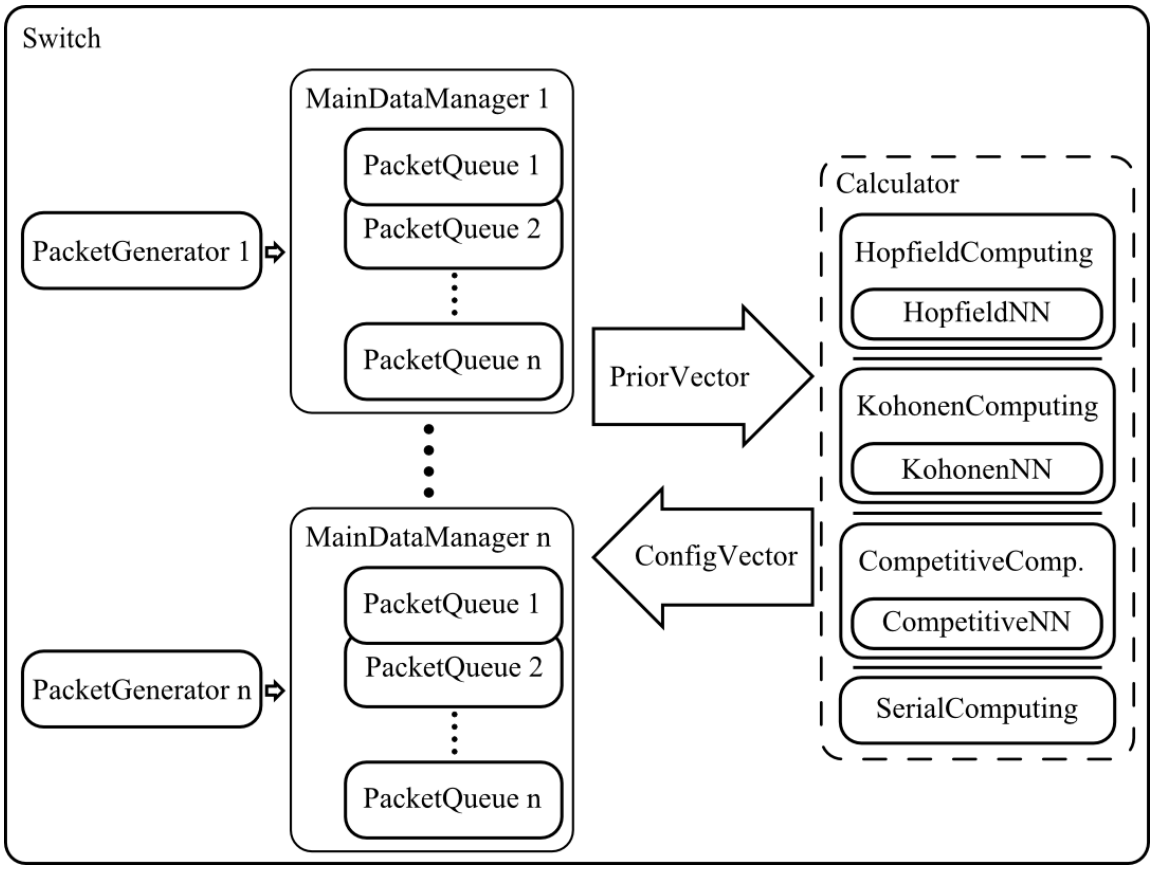
Simulační program má jednoduché rozhraní (obrázek *4.1*). V dolní části lze nastavit počet paketů pro měření času, délku paketu a prodlevu. Také je možné vypnout zobrazování aktuálních informací o zpracovávaných paketech (což je nutné pro objektivní měření), které se zobrazují v panelech nahoře. V levé části je výběr řídící neuronové sítě a počet vstupů přepínače. V pravé dolní části je výstup, ve kterém se zobrazují informace o proběhnuté simulaci. Simulace se spouští tlačítkem „START“ a ukončuje se tlačítkem „STOP“. Tlačítko „x START“ s přilehlým textovým vstupem slouží ke spuštění opakovaného měření. Výsledný výstup je potom zapsán do souboru, který se vytvoří v adresáři modelu přepínače.



Obr. 4.1 Rozhraní naprogramovaného modelu

4.3 Jednotlivé programové jednotky

Vycházel jsem z modelu vypracovaném v [2]. Tento model byl vypracován v prostředí Matlab – Simulink a byl v něm zpracován čtyř–portový přepínač řízený Hopfieldovou neuronovou sítí. Pokud to bylo vhodné a možné držel jsem se při implementaci myšlenek z této práce a programové jednotky (objekty) odpovídají podsystémům z tohoto modelu.



Obr 4.2 Schéma objektové struktury

Třídy HopfieldComputing, KohonenComputing, CompetitiveComputing a SrialComputing jsou dědici třídy Calculator, který říká, jaké metody musí tyto objekty implementovat, aby byl objekt Switch schopný zajistit funkčnost simulace.

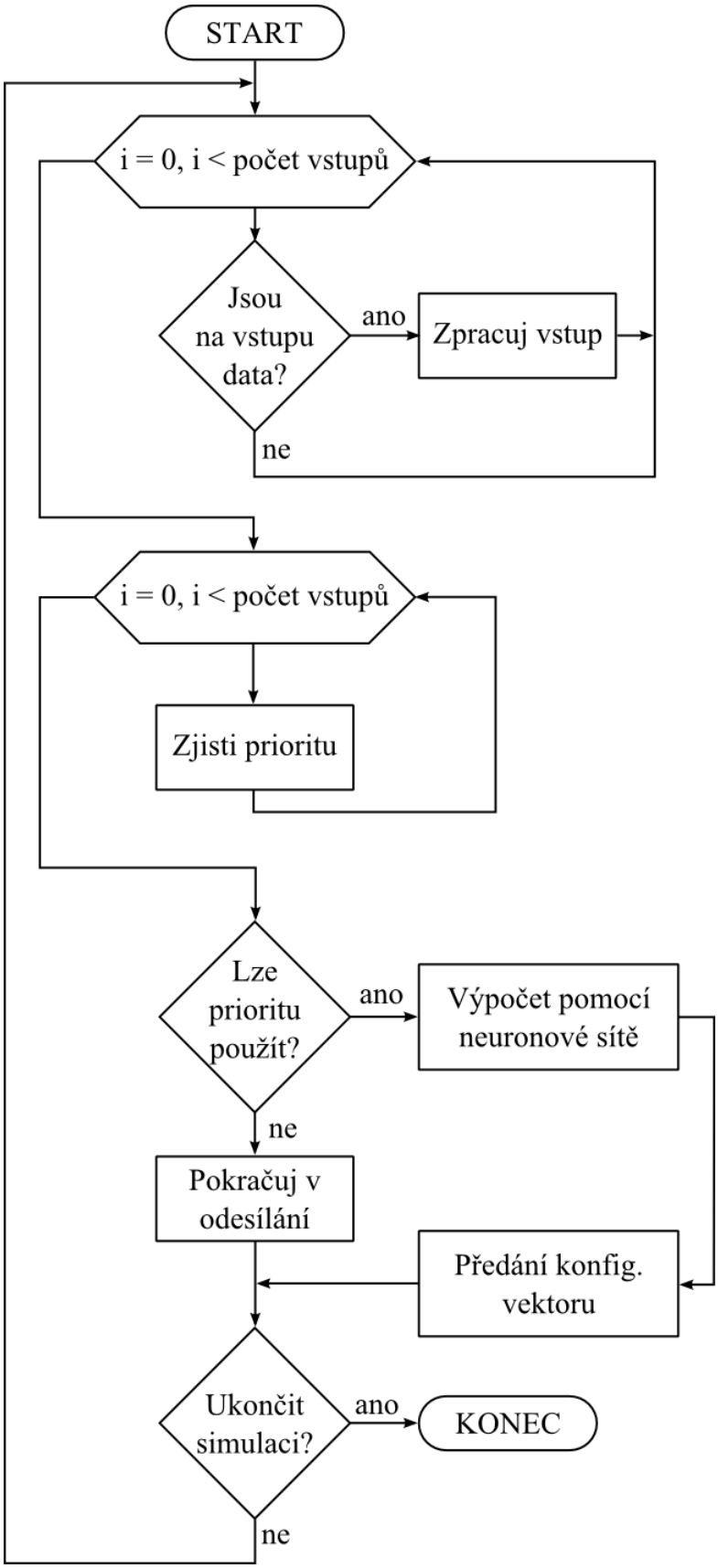
4.3.1 Hlavní správce simulace – třída Switch

Tento objekt je dědicem třídy TThread, která je implementací výpočtového vlákna. To umožňuje aplikaci zachovat funkční uživatelské rozhraní i v průběhu simulace, protože simulace běží v jiném vlákně než uživatelské rozhraní aplikace.

V hlavní smyčce vlákna, která běží tak dlouho, dokud nedostane povel k zastavení zvenčí, je napsána hlavní řídící logika přepínače. Zjednodušený vývojový diagram (neobsahuje příkazy pro výpis informací o simulaci) je prezentován na obrázku *4.3*.

V prvním cyklu je zkontrolováno, jestli jeden z objektů třídy PacketGenerator vygeneroval část paketu. Pokud ano, jsou data předána datovému manažeru (MainDataManager). Ten je buď zařadí do fronty (pokud už zná adresu příjemce), nebo je uchová ve své paměti a čeká, až dojdou další data, určující adresu příjemce paketu.

Druhý cyklus je určený k získání priorit z front na každém vstupu. Zároveň je předána logická hodnota, jestli je priorita použitelná (tzn. jestli na dané frontě náhodou neprobíhá odesílání). Pokud prioritu lze použít, dojde k předání prioritního vektoru (více v kapitole *4.4.1*) do výpočetního mechanismu neuronové sítě. Potom je předán konfigurační vektor z neuronové sítě všem datovým manažerům a v dalším průběhu cyklu dojde k odeslání paketu.



Obr. 4.3 Vývojový diagram hlavní smyčky simulace

4.3.2 Generátor paketů – třída PacketGenerator

Tato třída je zodpovědná za generování paketů pro simulaci. Generovaný paket má určité dané parametry, popsané v kapitole *2.3*. Délka paketu je nastavitelná v simulaci. Pakety jsou generovány s určitou prodlevou, která je také nastavitelná v simulaci. Pomocí těchto parametrů lze ovlivnit simulované zatížení přepínače.

V simulaci je přesně tolik instancí této třídy kolik je nastaveno vstupů přepínače.

4.3.3 Hlavní manažer dat – třída MainDataManager

Tato třída obhospodařuje přijímání dat z objektu PacketGenerator. Přijatá data ukládá do fronty PacketQueue která odpovídá cílové adrese v paketu. Pracuje také jako mezivrstva pro práci s frontou. A to hlavně v tom smyslu, že se stará o čtení priorit z prvního paketu ve frontě a umí interpretovat konfigurační vektor a tím přikázat určité frontě odeslání paketu.

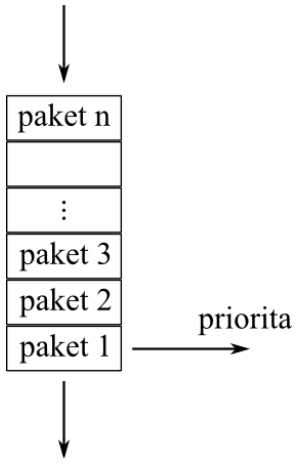
V simulaci se instance této třídy vyskytuje tolikrát jako je počet vstupních portů.

4.3.4 Fronta paketů – třída PacketQueue

Tato třída spravuje pakety, které dojdou do přepínače a jsou jí předány prostřednictvím nadřazené třídy MainDataManager. Také se stará o odstranění paketů vybraných k odeslání. Jedná se o upravenou frontu FIFO, která je schopná nedestruktivně číst prioritu z prvního paketu.

Dále jsou zde, pro účely simulace a měření výsledků, uchovávány statistiky přijatých a odeslaných paketů.

V simulaci se tento objekt vyskytuje v každé instanci třídy MainDataManager tolikrát, kolik je vstupních portů přepínače.



Obr 4.4 Fronta paketů

4.3.5 Obecný výpočet – třída Calculator

Tato třída obsahuje několik čistě virtuálních metod které jsou nutné pro správnou práci následujících tříd, které mají tuto jako předka. Je nutná proto abychom v instanci třídy Switch nemuseli přesně znát s kterým konkrétním výpočtovým aparátem pracujeme a mohli využít výhody objektového přístupu – polymorfismu.

Jedná se tedy především o metodu pro získání konfiguračního vektoru ze zadaného prioritního vektoru.

4.3.6 Výpočet Hopfieldovou neuronovou sítí – HopfieldComputing

Tato třída implementuje metody pro práci s Hopfieldovou neuronovou sítí a metody pro práci s dosaženými výsledky.

4.3.7 Výpočet Kohonenovou neuronovou sítí – KohonenComputing

Tato třída implementuje metody pro práci s Kohonenovou neuronovou sítí a metody pro práci s dosaženými výsledky.

4.3.8 Výpočet neuronovou sítí Competitive – CopetitiveComputing

Tato třída implementuje metody pro práci s neuronovou sítí Competitive a metody pro práci s dosaženými výsledky.

4.3.9 Obecná neuronová síť – třída NeuralNetwork

Podobně jako třída Calculator je třída NeuralNetwork určena pro sjednocení metod, které budou implementovány v konkrétních neuronových sítích.

Jsou to tedy metody pro učení, spuštění výpočtu a získání neinterpretovaného výstupu z neuronové sítě.

4.3.10 Výpočet Hopfieldovou sítí – třída HopfieldComputing

Třída implementuje algoritmus Hopfieldovy neuronové sítě popsaný v kapitole *3.2.4*. Také zde je implementován učící algoritmus přepsaný z prostředí Matlab – kapitola *3.2.3*.

4.3.11 Výpočet Kohonenovou sítí – třída KohonenComputing

Třída implementuje algoritmus Kohonenovy neuronové sítě popsaný v kapitole *3.3.4*. a nastavení vah.

4.3.12 Výpočet sítí Competitive – třída CompetitiveComputing

Třída implementuje algoritmus neuronové sítě Competitive popsaný v kapitole *3.4.4* a nastavení vah.

4.3.13 Sériový výpočet – třída SerialComputing

Tento blok slouží pouze pro porovnání správnosti výsledků a v počátku implementace sloužil jako jediný fungující rozhodovací mechanismus pro odesílání paketů. V podstatě implementuje numerické řešení optimalizačního problému z kapitoly *2.4*.

4.4 Komunikace mezi objekty

V obrázku *4.2* jsou znázorněny 2 hlavní informační kanály: PriorVector a ConfigVector.

4.4.1 Vektor priorit – PriorVector

Nejedná se až tolik o vektor, ale spíše o matici N × N (N = počet vstupních portů), která je až v pozdějším zpracování třídou Calculator (tedy některým z jejich konkrétních dědiců) převedena na vektor. Hodnoty v této datové struktuře jsou všechny hodnoty priorit odečtených ze všech front všech vstupů, které jsou určeny pro optimalizační výpočet.

4.4.2 Konfigurační vektor – ConfigVector

Je to jednorozměrné pole, jehož hodnoty jsou upraveny tak, aby jim rozuměl objekt MainDataManager – hodnota 1 značí odeslání paketu a hodnota -1 značí opak. Tyto hodnoty jsou získány v třídě Calculator (tedy některým z jejich konkrétních dědiců) interpretací hodnot výstupu dané neuronové sítě.

4.5 Doplňkové třídy nesouvisející přímo se simulací

Třída PatternGenerator

Slouží pro tvorbu konfiguračních vzorů. Jejím vstupem je počet vstupních portů a hlavním výstupem je matice konfiguračních vektorů vhodných pro učení neuronových sítí.

Různé matematické knihovny

Slouží hlavně pro výpočet vah Hopfieldovy neuronové sítě, který je realizován algoritmem přepsaným z prostředí Matlab – Simulink z kapitoly *3.2.3*. Výpočet vah pro Hopfieldovu neuronovou síť je nejsložitější v celé simulaci, také trvá poměrně dlouho a určitě nelze implementovat přímo v síťovém přepínači. Váhy tedy musí být vypočteny na počítači a do přepínače napevno nastaveny.

Výpočet bohužel selhává pro nastavení vah přepínače s osmi vstupy. Je to pravděpodobně způsobeno jeho vysokými paměťovými nároky.

5 POROVNÁNÍ VÝSLEDKŮ

Pro porovnání výsledků jsem zvolil dvě kritéria: čas a počet operací (iterací výpočtu). Měření doby výpočtu neuronovou sítí není příliš dobré kritérium, protože se jedná o paralelní strukturu a na osobním počítači s jedním procesorem proto nejsme schopni dosáhnout jejího plného potenciálu. Navíc neuronová síť provádí vzhledem ke své větší složitosti oproti sériovému výpočtu i více operací.

Výsledky jsem měřil i pro různé simulované zatížení přepínače a pro různý počet vstupních portů. Testy byly prováděny na pracovní stanici s procesorem AMD Athlon 1133MHz a 512 MB RAM. Jelikož v některých konfiguracích (počet vstupů × typ neuronové sítě) nebylo možné buď provést naučení sítě nebo síť pracovala příliš pomalu (z důvodů časové nebo prostorové složitosti algoritmů) jsou některé výsledky nahrazeny náhradní hodnotou (30000 ms).

Výsledky byly měřeny 12× a po odstranění nejmenší a největší hodnoty z nich byl vytvořen aritmetický průměr.

5.1 Počty konfiguračních vzorů

Pro lepší představu o optimalizačním problému uvedu tabulku s počty konfiguračních vzorů. V tabulce je vidět kolik možných konfigurací (výstupní port × zdrojová fronta), je pro daný počet vstupů. Počty konfigurací rostou s faktoriálem počtu vstupů *n!*. V případě neuronových sítí Competitive a Kohonen jsou tedy počty výstupních neuronů dost velké, (jelikož sítě mají pro každý vzor jeden neuron), naproti tomu Hopfieldova síť má počet neuronů závislý na druhé mocnině počtu vstupů n^2.

Počet vstupů *n*	**Počet vzorů**	***n*²**
2	2	4
3	6	9
4	24	16
5	120	25
6	720	36
7	5040	49
8	40320	64

Tabulka 5.1 Počty konfiguračních vzorů

5.2 Porovnání sítí mezi sebou

Nejprve porovnáme neuronové sítě navzájem. Hodnoty jsou měřeny pro výchozí hodnoty (prodleva maximálně 20 cyklů a délka paketu maximálně 20 hodnot) a měřen byl čas potřebný k odeslání 5000 paketů.

	Počet vstupů						
Typ sítě	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Competitive	232	503	8423	30000*	30000*	30000*	30000*
Hopfield	1508	1642	2994	5301	9851	15203	30000**
Kohonen	218	232	256	425	1592	15406	30000*

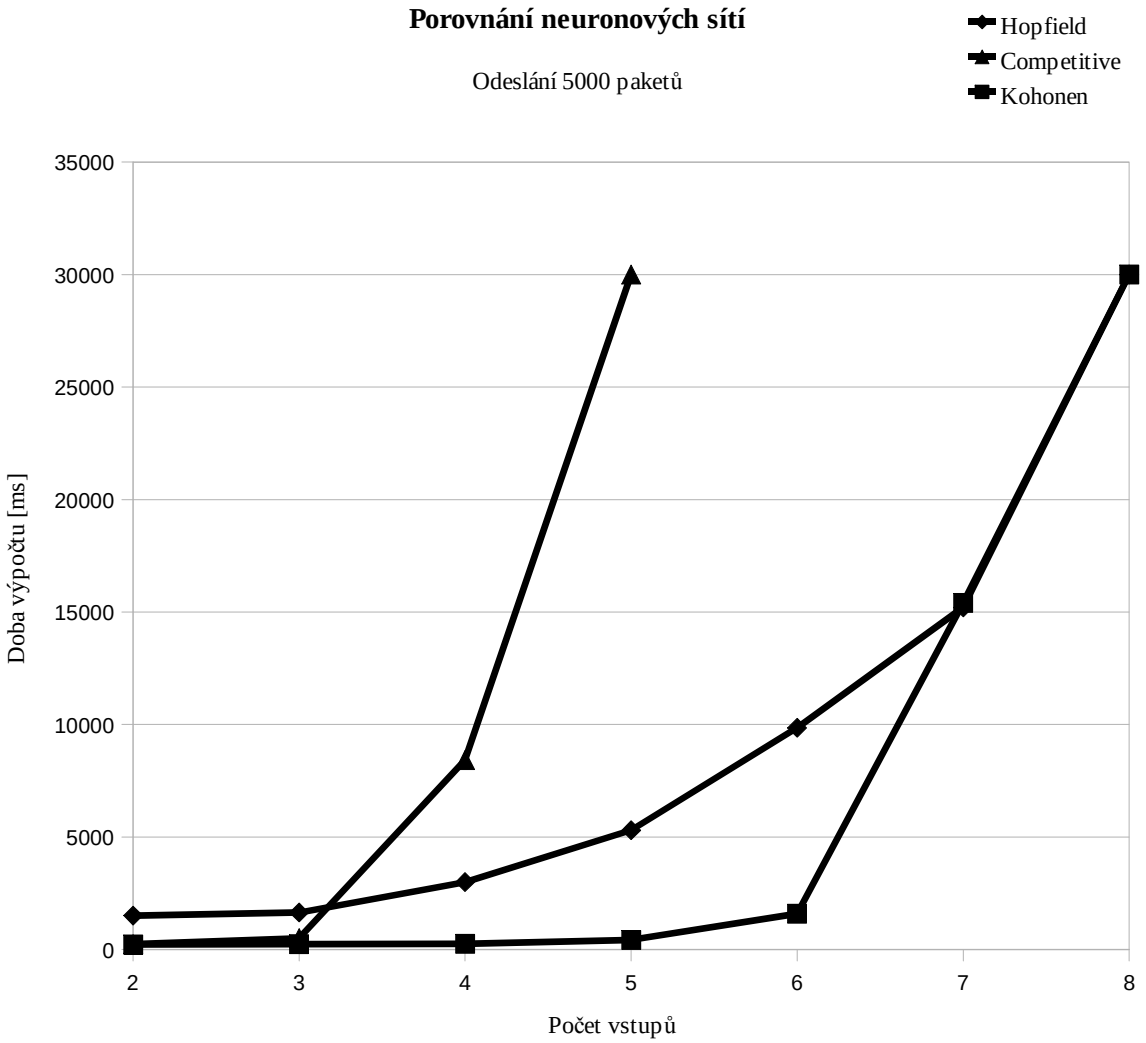
Tabulka 5.2 Doba výpočtu [ms]

* = výpočet trval příliš dlouho
** = selhal učící algoritmus neuronové sítě

Graficky jsou hodnoty z této tabulky znázorněny v grafu na obrázku *5.1*. Rozdíl mezi neuronovými sítěmi je vidět na první pohled. Zatímco Hopfieldova síť má horší výsledky na menším počtu vstupů, tak ostatní 2 sítě začínají mít problémy na větším množství vstupů. Tento rozdíl plyne z faktu, že Hopfieldova síť má jen jednu vrstvu a počet neuronů neroste s faktoriálem ale jen s druhou mocninou. To je také vidět na grafu. Hopfieldova síť se zpomaluje pomaleji než ostatní dvě.

Horší výsledky Hopfieldovy sítě na menším počtu vstupů jsou naopak způsobeny její rekurentní topologií (tedy tím, že potřebuje k získání výsledku více iterací).

Neuronová síť Competitive má bohužel výsledky úplně nejhorší protože kombinuje dopředný i rekurentní výpočet a navíc jí rychle přibývá počet neuronů a u většího množství vstupů se to negativně projevuje.



Obr. 5.1 Graf závislosti doby výpočtu na počtu vstupů

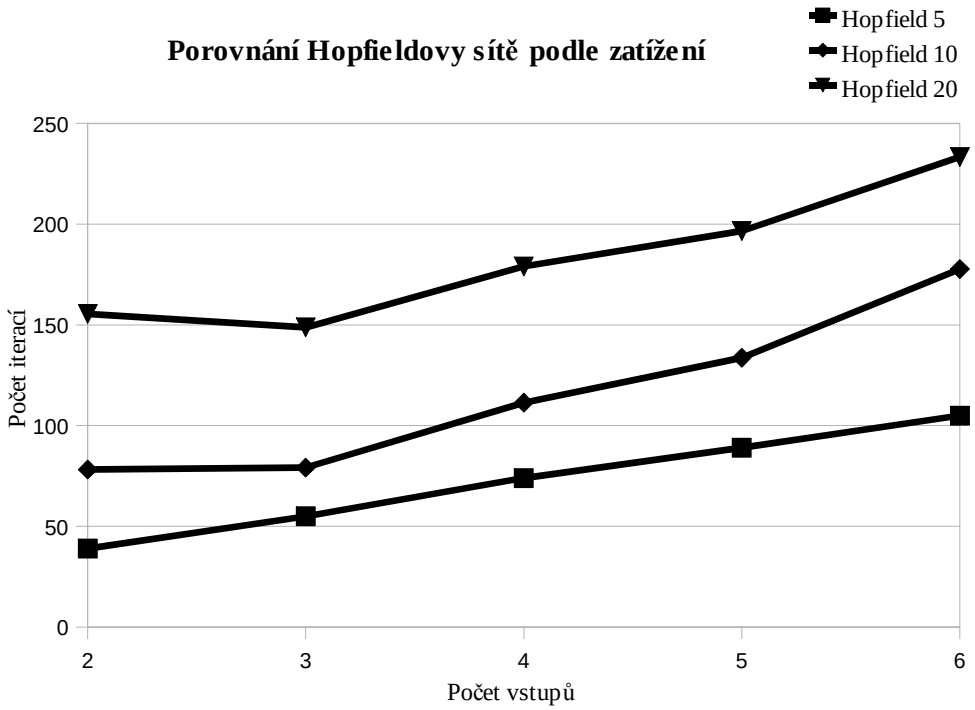
Tabulka *5.3* porovnává neuronové sítě podle počtu potřebných iterací pro dokončení optimalizačního výpočtu. Chybějící výsledky mají stejnou příčinu jako v tabulce *5.2*. U neuronové sítě Competitive je vidět prudký nárůst iterací s rostoucím počtem vstupů. To způsobuje velkou časovou složitost výpočtu. Kohonenova síť je pouze dopředná a proto má vždy jen jednu iteraci.

	Počet vstupů						
Typ sítě	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Hopfield	155	148	179	196	233	248	–
Kohonen	1	1	1	1	1	1	1
Competitive	8	62	263	–	–	–	–

Tab 5.3 Počet iterací výpočtu

5.3 Porovnání Hopfieldovy sítě pro různé zatížení

Zajímavým jevem u Hopfieldovy sítě byla závislost počtu iterací na zatížení modelu. Čím více front obsahovalo alespoň nějaký paket (vstupní hodnoty byly odlišné od krajních hodnot -1 a 1), tím rychleji byla síť schopna konvergovat k nějakému konfiguračnímu vzoru. Na obrázku *5.2* jsou v grafu zobrazeny závislosti počtu iterací na zatížení neuronové sítě. Hodnoty 5, 10 a 20 znamenají maximální možnou prodlevu pro generování paketů nastavenou v simulaci.

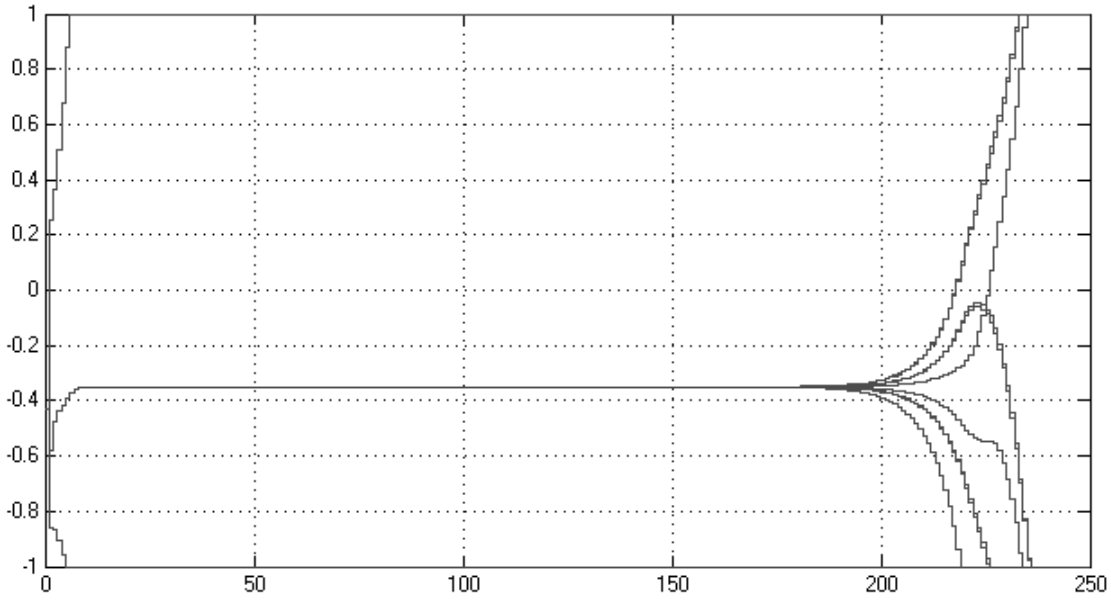


Obr 5.2 Graf počtu iterací v závislosti na zatížení Hopfieldovy sítě

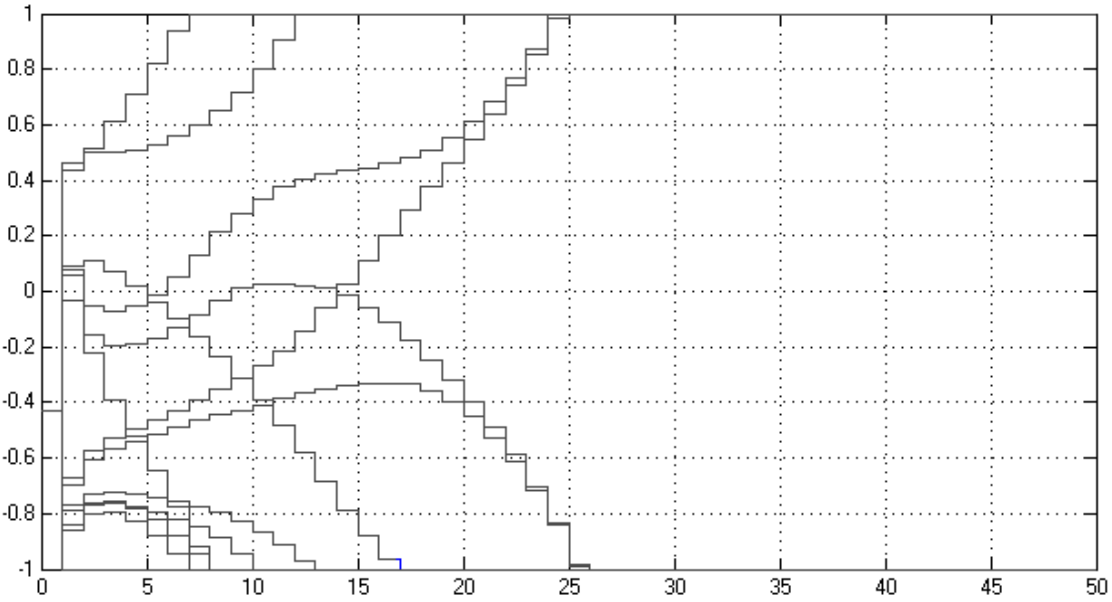
Obrázky *5.3* a *5.4* zobrazují průběh konvergence 16–ti neuronové Hopfieldovy sítě pro dva odlišné konfigurační vektory. Obrázky pochází z prostředí Matlab – Simulink, ve kterém lze vytvořit stejnou neuronovou síť jako v mém simulačním programu (díky stejnému učícímu algoritmu).

Na prvním obrázku je konfigurační vektor složen převážně z hodnot -1 a je vidět, že Hopfieldova síť došla k výsledku až po téměř 250–ti iteracích. Na obrázku *5.4* je konfigurační vektor upraven, aby obsahoval více hodnot z intervalu (-1, 1) a je vidět, že k výsledku síť došla už po necelých 30–ti iteracích.

Z obrázku *5.3* se dá odvodit, že by nebylo vhodné omezovat maximální počet iterací, protože síť začíná konvergovat až po nějakém počtu kroků a do té doby je její výsledek nehodnotný a nešel by ani rozumně interpretovat (například hodnoty větší než určitá hodnota zaokrouhlit na 1).



Obr 5.3 Konvergence Hopfieldovy sítě pro malé zatížení



Obr 5.4 Konvergence Hopfieldovy sítě pro větší zatížení

5.4 Porovnání správnosti výsledků

V některých případech se stane, že neuronová síť vybere jiný konfigurační vektor než sériový optimalizační výpočet. Ve většině případů jde pouze o ty kombinace, které mají více stejně ohodnocených řešení a sériový výpočet vždy bere první nejlepší.

Naproti tomu neuronová síť může zvolit náhodné řešení, pokud je více ekvivalentních. V případě Hopfieldovy sítě však v mizivém procentu případů dojde k výběru neoptimálního konfiguračního vzoru.

Výsledky měření pro odeslání 5000 paketů na modelu přepínače se čtyřmi vstupy jsou v tabulce *5.4*.

Typ sítě	**Odlišné vzory [%]**	**Chybné vzory [%]**
Hopfield	19,9	0,03
Kohonen	1,38	0
Competitive	19,01	0

Tabulka 5.4 Chyby výpočtu

6 ZÁVĚR

V diplomové práci jsem se věnoval optimalizaci aktivního síťového prvku pomocí neuronové sítě. V průběhu řešení jsem vyzkoušel různé typy neuronových sítí, které by bylo možné použít pro řízení funkce přepínače. Některé zvolené sítě nebyly vhodné, podle dostupných zdrojů [13], vůbec (např. *MLP – Multi–layer perceptron*) a proto jsem je ani neuváděl v této práci. Jako vhodné se nakonec zdají hlavně síť Kohonenova a Hopfieldova, která byla použita už v předchozím zpracování tohoto úkolu. Jako poměrně nevhodná se jeví síť Competitive, jelikož její výpočet trvá poměrně dlouho a nezaručuje, že pokaždé dojde alespoň k nějakému výsledku.

Ve výsledném zhodnocení jsem porovnal navzájem neuronové sítě podle různých kritérií. Z výsledků plyne, že nejvhodnější paralelní struktura je pravděpodobně Kohonenova neuronová síť, která je nejrychlejší pro řízení síťového prvku se čtyřmi vstupy. Pro větší počet vstupů by pravděpodobně byla vhodnější Hopfieldova neuronová síť, jelikož počet neuronů (výpočetních jednotek) u ní neroste takovým tempem jako u ostatních sítí a tím si zachovává realizovatelnost.

Velká výhoda Kohonenovy sítě spočívá v dopředné architektuře a tím pádem v rychlém (jednoprůchodovém) výpočtu. Na druhou stranu je v každém jejím neuronu realizována složitější výpočetní funkce, která se neskládá jen z násobení a sčítání. Kohonenova síť vyhodnocuje euklidovskou vzdálenost vzoru od aktuálního stavu, a proto potřebuje funkce jako např. druhá mocnina a odmocnina. Dobrou zprávou je, že umocňování čísel lze dosáhnout násobením a odmocnina není potřebná, jelikož nám nejde o přesnou vzdálenost, ale chceme pouze znát nejmenší hodnotu ze všech výpočtů.

Hopfieldova síť je pro problémy řízení síťových přepínačů využívána tradičně, ale narazil jsem hlavně na případy, kdy neohodnocujeme priority paketů. Pokud do této sítě vstupují nebinární hodnoty, stává se, že jí trvá poměrně dlouho než dojde k výsledku.

Vytvořený programový model je koncipován tak, aby nebyl problém do něj začlenit další typy neuronových sítí. Tím je vhodný pro případné další testování. Je naprogramovaný ve známém a volně dostupném prostředí.

7 SEZNAM POUŽITÉ LITERATURY

[1]Roupec J. Počítačové sítě [Dokument] VUT: 2002

[2]Pokorný P. Návrh síťového prvku pomocí neuronové sítě [Dokument] VUT: 2008

[3]Mathworks, Neural Network Toolbox [Online] Dostupné z <http://www.mathworks.com/access/helpdesk/help/pdf_doc/nnet/nnet.pdf>

[4]Malasri K., Hopfield Network Applet [Online] Dostupné z <http://www.cbu.edu/~pong/ai/hopfield/hopfieldapplet.html>

[5]Rojas R. Neural Networks Springer-Verlag Berlin : 1996

[6]Heaton Research, The Hopfield Neural Network [Online] Dostupné z <http://www.heatonresearch.com/articles/2/page5.html>

[7]Davey N., Hunt S.P., Adams R.G. High Capacity Recurrent Associative Memories [Dokument] University of Hertfordshire: 2004

[8]Honkela T., SOM related references [Online] Dostupné z <http://mlab.taik.fi/~timo/som/references.html>

[9]Germano T., Self-Organizing Maps [Online] Dostupné z <http://davis.wpi.edu/~matt/courses/soms/>

[10]Neural Networks, Simple competitive networks [Online] Dostupné z <http://cse.stanford.edu/class/sophomore-college/projects-00/neural-networks/Architecture/competitive.html>

[11]Rapisarda P., Competitive and recursive networks [Online] Dostupné z <http://www.fdaw.unimaas.nl/education/34_NeuralNetworks/Lecture8Slides.pdf>

[12]Borland, Turbo Explorer Home Page [Online] Dostupné z <http://www.turboexplorer.com/>

[13]Šťastný,J., Škorpil,V: Neural Networks Learning Methods Comparison , International Journal WSEAS Transactions on Circuits and Systems, Issue 4, Volume 4 April 2005, ISSN 1109-2734, pp. 325-330